

Separation Logic 2/4

Jean-Marie Madiot

Inria Paris

January 31, 2022

slides (mostly) from Arthur Charguéraud

Chapter 7

The Frame Rule

Preservation of independent state

We have:

$$\{r \mapsto 2\} (\text{incr } r) \{\lambda_. r \mapsto 3\}$$

We also have:

$$\{r \mapsto 2 * s \mapsto 7\} (\text{incr } r) \{\lambda_. r \mapsto 3 * s \mapsto 7\}$$

Preservation of independent state

We have:

$$\{r \mapsto 2\} (\text{incr } r) \{\lambda_. r \mapsto 3\}$$

We also have:

$$\{r \mapsto 2 * s \mapsto 7\} (\text{incr } r) \{\lambda_. r \mapsto 3 * s \mapsto 7\}$$

More generally:

$$\{r \mapsto 2 * H\} (\text{incr } r) \{\lambda_. r \mapsto 3 * H\}$$

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:

$$\frac{\{H_1\} t \{\lambda x. H'_1\}}{\{H_1 * H_2\} t \{\lambda x. H'_1 * H_2\}}$$

Frame rule and allocation

We have:

$$\{\text{true}\} (\text{ref } 3) \{\lambda r. (r \mapsto 3)\}$$

By the frame rule, we have:

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

Frame rule and allocation

We have:

$$\{\text{true}\} (\text{ref } 3) \{\lambda r. (r \mapsto 3)\}$$

By the frame rule, we have:

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

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:

$$\forall pL. \{p \rightsquigarrow \text{MList } L\} (\text{mlength } p) \{ \lambda n. \text{'}n = \text{length } L\text{'} * p \rightsquigarrow \text{MList } L \}$$

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:

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

We prove this specification by induction on L .

Verification of mlength: nil case

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

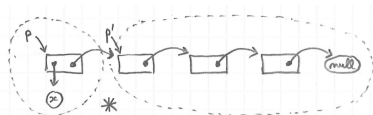
$$\{p \rightsquigarrow \text{MList nil}\} (0) \{\lambda n. \ulcorner n = \text{length nil} \urcorner * p \rightsquigarrow \text{MList nil}\}$$

Same as:

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

Verification of mlength: using the frame rule

$\forall Lp. \{p \rightsquigarrow \text{MList } L\} (\text{mlength } p) \{ \lambda n. \ulcorner n = \text{length } L \urcorner * p \rightsquigarrow \text{MList } L \}$



Assume $L = x :: L'$.

$p \rightsquigarrow \text{MList } L$	pre-condition
$p \rightsquigarrow \{ \text{hd}=x; \text{tl}=p' \} * p' \rightsquigarrow \text{MList } L'$	by unfolding
$p \rightsquigarrow \{ \text{hd}=x; \text{tl}=p' \} * p' \rightsquigarrow \text{MList } L' * \ulcorner n' = L' \urcorner$	frame +induction
$p \rightsquigarrow \text{MList } L * \ulcorner n' + 1 = x :: L' \urcorner$	by folding
$p \rightsquigarrow \text{MList } L * \ulcorner n = L \urcorner$	post-condition

Instantiation of the frame rule

Induction hypothesis:

$$\begin{aligned} & \{p' \rightsquigarrow \text{MList } L'\} \\ & (\text{mlength } p') \\ & \{\lambda n'. \ulcorner n = \text{length } L' \urcorner * p' \rightsquigarrow \text{MList } L'\} \end{aligned}$$

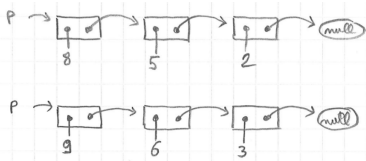
By the frame rule:

$$\begin{aligned} & \{p' \rightsquigarrow \text{MList } L' * p \rightsquigarrow \{\text{hd}=x; \text{tl}=p'\} \} \\ & (\text{mlength } p') \\ & \{\lambda n. \ulcorner n = \text{length } L' \urcorner * p' \rightsquigarrow \text{MList } L' * p \rightsquigarrow \{\text{hd}=x; \text{tl}=p'\} \} \end{aligned}$$

Verification of mlength: coq

Coq demo.

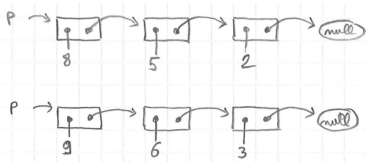
Verification of in-place increment



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

$$\forall pL. \{p \rightsquigarrow \text{MList } L\} (\text{list_incr } p) \{\lambda.. p \rightsquigarrow \text{MList}(\text{map}(+1) L)\}$$

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
```

$$\forall pL. \{p \rightsquigarrow \text{MList } L\} (\text{list_incr } p) \{\lambda_. p \rightsquigarrow \text{MList}(\text{map } (+1) L)\}$$

Exercise: describe how the frame rule is used for in-place increment.

Verification of in-place increment: frame rule

$\forall pL. \{p \rightsquigarrow \text{MList } L\} (\text{list_incr } p) \{\lambda_. p \rightsquigarrow \text{MList } (\text{map } (+1) L)\}$

Assume $L = x :: L'$.

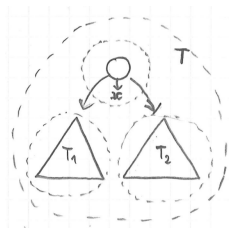
$p \rightsquigarrow \text{MList } L$		pre-condition
$p \rightsquigarrow \{\text{hd}=x; \text{tl}=p'\}$	$* p' \rightsquigarrow \text{MList } L'$	by unfolding
$p \rightsquigarrow \{\text{hd}=x + 1; \text{tl}=p'\}$	$* p' \rightsquigarrow \text{MList } L'$	incrementing
$p \rightsquigarrow \{\text{hd}=x +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

Verification of list_incr: coq

Coq.

Specification of tree copy

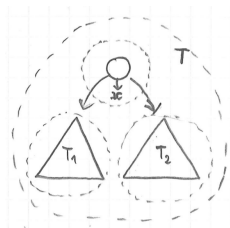
```
let rec copy (p:node) : node =  
  if p == null then null else  
  let p1' = copy p.left in  
  let p2' = copy p.right in  
  { item = p.item;  
    left = p1';  
    right = p2' }
```



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' }
```

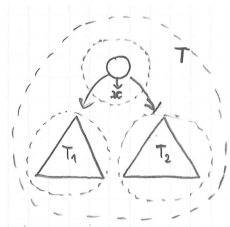


Exercise: specify the tree copy function.

$$\forall pT. \{p \rightsquigarrow \text{Mtree } T\} (\text{copy } p) \{\lambda p'. p \rightsquigarrow \text{Mtree } T * p' \rightsquigarrow \text{Mtree } T\}$$

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.

$$\forall pT. \{p \rightsquigarrow \text{Mtree } T\} (\text{copy } p) \{\lambda p'. p \rightsquigarrow \text{Mtree } T * p' \rightsquigarrow \text{Mtree } T\}$$

Exercise: describe how the frame rule is used for tree copy.

Verification of tree copy: frame rule

$p \rightsquigarrow \text{Mtree } T$	by pre-condition
$\underline{p \mapsto \{x; p_1; p_2\}} * p_1 \rightsquigarrow \text{Mtree } T_1 * \underline{p_2 \rightsquigarrow \text{Mtree } T_2}$	by unfolding
$p \mapsto \underline{\{x; p_1; p_2\}} * \underline{p_1 \rightsquigarrow \text{Mtree } T_1} * p_2 \rightsquigarrow \text{Mtree } T_2$ $* \underline{p'_1 \rightsquigarrow \text{Mtree } T_1}$	<u>frame</u> +induction
$p \mapsto \{x; p_1; p_2\} * p_1 \rightsquigarrow \text{Mtree } T_1 * p_2 \rightsquigarrow \text{Mtree } T_2$ $* p'_1 \rightsquigarrow \text{Mtree } T_1 * p'_2 \rightsquigarrow \text{Mtree } T_2$	<u>frame</u> +induction
$p \mapsto \{x; p_1; p_2\} * p_1 \rightsquigarrow \text{Mtree } T_1 * p_2 \rightsquigarrow \text{Mtree } T_2$ $* p' \mapsto \{x; p'_1; p'_2\} * p'_1 \rightsquigarrow \text{Mtree } T_1 * p'_2 \rightsquigarrow \text{Mtree } T_2$	by allocation
$p \rightsquigarrow \text{Mtree } T$ $* p' \rightsquigarrow \text{Mtree } T$	by folding

Summary

$$\frac{\{H_1\} t \{\lambda x. H'_1\}}{\{H_1 * H_2\} t \{\lambda x. H'_1 * H_2\}} \text{FRAME}$$

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

$p \rightsquigarrow \text{MList } L$

$p \rightsquigarrow \{\text{hd}=x; \text{tl}=p'\} * p' \rightsquigarrow \text{MList } L'$ by unfolding

$p \rightsquigarrow \{\text{hd}=x + 1; \text{tl}=p'\} * p' \rightsquigarrow \text{MList } L'$ incrementing

$p' \rightsquigarrow \text{MList } L'$ frame begins

$p' \rightsquigarrow \text{MList } (\text{map } (+1) L')$ by induction

$p \rightsquigarrow \{\text{hd}=x + 1; \text{tl}=p'\} * p' \rightsquigarrow \text{MList } (\text{map } (+1) L')$ frame ends

$p \rightsquigarrow \text{MList } ((x + 1) :: (\text{map } (+1) L'))$ by folding

$p \rightsquigarrow \text{MList } (\text{map } (+1) L)$ by rewriting

Chapter 8

Small footprint specifications

Small footprint access to records

$$p \rightsquigarrow \{\text{hd}=v; \text{tl}=q\} \equiv p.\text{hd} \mapsto v * p.\text{tl} \mapsto q$$

Specification of a write on the head field:

$$\{p \rightsquigarrow \{\text{hd}=w; \text{tl}=q\}\} (p.\text{hd} \leftarrow v) \{\lambda_. p \rightsquigarrow \{\text{hd}=v; \text{tl}=q\}\}$$

Small footprint access to records

$$p \rightsquigarrow \{\text{hd}=v; \text{tl}=q\} \equiv p.\text{hd} \mapsto v * p.\text{tl} \mapsto q$$

Specification of a write on the head field:

$$\{p \rightsquigarrow \{\text{hd}=w; \text{tl}=q\}\} (p.\text{hd} \leftarrow v) \{\lambda_. p \rightsquigarrow \{\text{hd}=v; \text{tl}=q\}\}$$

Same, but with a smaller footprint:

$$\{p.\text{hd} \mapsto w\} (p.\text{hd} \leftarrow v) \{\lambda_. p.\text{hd} \mapsto v\}$$

or $\{p.\text{hd} \mapsto -\} (p.\text{hd} \leftarrow v) \{\lambda_. p.\text{hd} \mapsto v\}$

Small footprint access to records

$$p \rightsquigarrow \{\text{hd}=v; \text{tl}=q\} \equiv p.\text{hd} \mapsto v * p.\text{tl} \mapsto q$$

Specification of a write on the head field:

$$\{p \rightsquigarrow \{\text{hd}=w; \text{tl}=q\}\} (p.\text{hd} \leftarrow v) \{\lambda_. p \rightsquigarrow \{\text{hd}=v; \text{tl}=q\}\}$$

Same, but with a smaller footprint:

$$\begin{aligned} & \{p.\text{hd} \mapsto w\} (p.\text{hd} \leftarrow v) \{\lambda_. p.\text{hd} \mapsto v\} \\ \text{or } & \{p.\text{hd} \mapsto -\} (p.\text{hd} \leftarrow v) \{\lambda_. p.\text{hd} \mapsto v\} \end{aligned}$$

From small to large footprint using frame:

$$\{p.\text{hd} \mapsto w * p.\text{tl} \mapsto q\} (p.\text{hd} \leftarrow v) \{\lambda_. p.\text{hd} \mapsto v * p.\text{tl} \mapsto q\}$$

Representation predicate for arrays

Representation predicate for C arrays:

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

where:

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

Representation predicate for arrays

Representation predicate for C arrays:

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

where:

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

Representation predicate for ML arrays:

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

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

Small footprint specifications of array operations

$i \in \text{dom } L \Rightarrow$

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

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

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

Exercise: give small footprint specifications for array operations.

How to derive the large footprint specifications from them?

Small footprint specifications of array operations

$i \in \text{dom } L \Rightarrow$

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

$\{p \rightsquigarrow \text{Array } L\} (\text{Array.set } i \ p \ v) \{\lambda_. p \rightsquigarrow \text{Array } (L[i := v])\}$

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

Exercise: give small footprint specifications for array operations.

How to derive the large footprint specifications from them?

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

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

$\{p[i] \mapsto -\} (\text{Array.set } i \ p \ v) \{\lambda_. p[i] \mapsto v\}$

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

Small footprint specifications of array operations

$i \in \text{dom } L \Rightarrow$

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

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

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

Exercise: give small footprint specifications for array operations.

How to derive the large footprint specifications from them?

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

$\{p[i] \mapsto v\} \quad (\text{Array.get } i \ p) \quad \{\lambda x. \ulcorner x = v \urcorner * p[i] \mapsto v\}$

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

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

$$\begin{aligned} p \rightsquigarrow \text{Array } L &= p[i] \mapsto L[i] \\ &* p.\text{length} \mapsto |L| \\ &* \bigotimes_{v \text{ at index } j \text{ in } L} p[j] \mapsto v \\ &\quad \text{with } j \neq i \end{aligned}$$

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:

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

Access to a memory cell

In C, record and array accesses are treated uniformly:

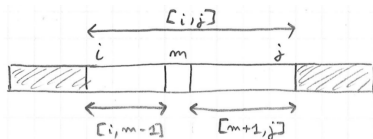
$p \rightarrow \text{hd} = v$	compiles to	$*(p + \text{hd}) = v$
$p[i] = v$	compiles to	$*(p + i) = v$
$i[p] = v$	compiles to	$*(i + p) = v \quad (\dots)$

Common small footprint specification for accessing a memory cell:

$$\{p \mapsto -\} (*p = v) \quad \{\lambda_. p \mapsto v\}$$
$$\{p \mapsto v\} (*p) \quad \{\lambda x. \ulcorner x = v \urcorner * p \mapsto v\}$$

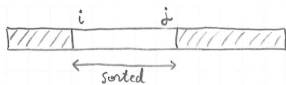
All other specifications for read and write operations are derivable.

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



$$\begin{aligned} \forall p L i j. \quad & 0 \leq i \leq j < |L| \Rightarrow \\ & \{ p \rightsquigarrow \text{Array } L \} \\ & (\text{qsort } i \ j \ p) \\ & \{ \lambda _. \exists L'. \quad (p \rightsquigarrow \text{Array } L') \quad \} \\ & * \text{sortedSeg } i \ j \ L' \\ & * \text{permut } L \ L' \\ & * \text{'}\forall a \in [0, i) \cup (j, |L|). L'[a] = L[a]\text{'} \end{aligned}$$

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

Group of array cells

Definition

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

where M has type “map int A ”, for some A .

Group of array cells

Definition

$$p \rightsquigarrow \text{Cells } M \equiv \bigotimes_{(i,v) \in M} p[i] \mapsto v$$

where M has type “map int A ”, for some A .

Conversions:

$$p \rightsquigarrow \text{Array } L = p.\text{length} \mapsto |L| * p \rightsquigarrow \text{Cells} (\text{mapOfList } L)$$

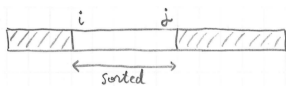
$$p \rightsquigarrow \text{Cells } \emptyset = \top$$

$$p \rightsquigarrow \text{Cells } \{(i, v)\} = p[i] \mapsto v$$

$$p \rightsquigarrow \text{Cells } (M_1 \uplus M_2) = p \rightsquigarrow \text{Cells } M_1 * p \rightsquigarrow \text{Cells } M_2 \quad (\text{when } M_1 \perp M_2)$$

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

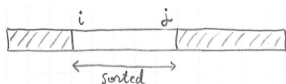
Small-footprint specification of quicksort



Exercise: give a small-footprint specification for quicksort.

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

Small-footprint specification of quicksort

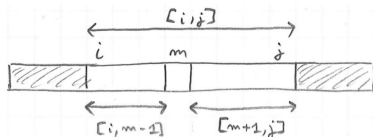


Exercise: give a small-footprint specification for quicksort.

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

$$\begin{aligned} \forall p M i j. \quad \text{dom } M = [i, j] \Rightarrow \\ & \{p \rightsquigarrow \text{Cells } M\} \\ & (\text{qsort } i \ j \ p) \\ & \{ \lambda_. \exists M'. \quad p \rightsquigarrow \text{Cells } M' \quad \} \\ & \quad * \ulcorner \text{sortedSeg } i \ j \ M' \urcorner \\ & \quad * \ulcorner \text{permut } M \ M' \urcorner \end{aligned}$$

Segment splitting in Quicksort



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

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

Note that:

$$p \rightsquigarrow \text{Cells } (M|_{[m, m]}) = p[m] \mapsto M[m]$$

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:

$i \in \text{dom } M \Rightarrow$

$\{p \rightsquigarrow \text{Cells } M\} (\text{Array.get } i \ p) \{\lambda x. \ulcorner x = M[i] \urcorner * p \rightsquigarrow \text{Cells } M\}$

$\{p \rightsquigarrow \text{Cells } M\} (\text{Array.set } i \ p \ v) \{\lambda_. p \rightsquigarrow \text{Cells } (M[i := v])\}$

Summary

Representation of a full array using a list:

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

Small footprint accesses:

$$\begin{aligned} \{p[i] \mapsto v\} & \quad (\text{Array.get } i \text{ p}) \quad \{\lambda_. p[i] \mapsto v\} \\ \{p[i] \mapsto -\} & \quad (\text{Array.set } i \text{ p } v) \quad \{\lambda x. \ulcorner x = v \urcorner * p[i] \mapsto v\} \\ \{p.\text{length} \mapsto n\} & \quad (\text{Array.length } p) \quad \{\lambda x. \ulcorner x = n \urcorner * p.\text{length} \mapsto n\} \end{aligned}$$

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

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

Chapter 9

Heap entailment

Definition of Hprop

Let:

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

For example:

$$\ulcorner \urcorner \quad : \quad \text{Hprop}$$

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

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

In particular:

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

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: $H * \top = H$

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

Extrusion of pure facts: $(\top * H) m = P \wedge (H m)$

Extensionality

Functional extensionality:

$$(\forall x. f x = g x) \Rightarrow (f = g) \quad (\text{if } f, g: A \rightarrow B)$$

Propositional extensionality:

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

Predicate extensionality:

$$(\forall x. P x \Leftrightarrow Q x) \Rightarrow (P = Q) \quad (\text{if } P, Q: A \rightarrow \text{Prop})$$

Heap predicate extensionality:

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

Heap entailment

Definition:

$$H_1 \triangleright H_2 \quad \equiv \quad \forall m. H_1 m \Rightarrow H_2 m$$

For example:

$$(r \mapsto 6) \triangleright \exists n. (r \mapsto n) * \text{'even } n\text{'}$$

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

Heap entailment as a partial order

$$H_1 \triangleright H_2 \quad \equiv \quad \forall m. H_1 m \Rightarrow H_2 m$$

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

REFLEXIVITY

$$\frac{}{H \triangleright H}$$

TRANSITIVITY

$$\frac{H_1 \triangleright H_2 \quad H_2 \triangleright H_3}{H_1 \triangleright H_3}$$

ANTISYMMETRY

$$\frac{H_1 \triangleright H_2 \quad H_2 \triangleright H_1}{H_1 = H_2}$$

Frame property for heap entailment

$$\frac{H_1 \triangleright H'_1}{H_1 * H_2 \triangleright H'_1 * H_2} \text{ENTAIL-FRAME}$$

For example, to prove:

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

it suffices to prove:

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

Heap implications: true or false?

1. $(r \mapsto 3) * (s \mapsto 4) \triangleright (s \mapsto 4) * (r \mapsto 3)$
2. $(r \mapsto 3) \triangleright (s \mapsto 4) * (r \mapsto 3)$
3. $(s \mapsto 4) * (r \mapsto 3) \triangleright (r \mapsto 4)$
4. $(s \mapsto 4) * (r \mapsto 3) \triangleright (r \mapsto 3)$
5. $\text{「False」} * (r \mapsto 3) \triangleright (s \mapsto 4) * (r \mapsto 4)$
6. $(r \mapsto 4) * (s \mapsto 3) \triangleright \text{「False」}$
7. $(r \mapsto 4) * (r \mapsto 3) \triangleright \text{「False」}$
8. $(r \mapsto 3) * (r \mapsto 3) \triangleright \text{「False」}$

Heap implications: true or false?

1. $(r \mapsto 3) * (s \mapsto 4) \triangleright (s \mapsto 4) * (r \mapsto 3)$ true
2. $(r \mapsto 3) \triangleright (s \mapsto 4) * (r \mapsto 3)$
3. $(s \mapsto 4) * (r \mapsto 3) \triangleright (r \mapsto 4)$
4. $(s \mapsto 4) * (r \mapsto 3) \triangleright (r \mapsto 3)$
5. $\text{'False'} * (r \mapsto 3) \triangleright (s \mapsto 4) * (r \mapsto 4)$
6. $(r \mapsto 4) * (s \mapsto 3) \triangleright \text{'False'}$
7. $(r \mapsto 4) * (r \mapsto 3) \triangleright \text{'False'}$
8. $(r \mapsto 3) * (r \mapsto 3) \triangleright \text{'False'}$

Heap implications: true or false?

1. $(r \mapsto 3) * (s \mapsto 4) \triangleright (s \mapsto 4) * (r \mapsto 3)$ true
2. $(r \mapsto 3) \triangleright (s \mapsto 4) * (r \mapsto 3)$ false
3. $(s \mapsto 4) * (r \mapsto 3) \triangleright (r \mapsto 4)$
4. $(s \mapsto 4) * (r \mapsto 3) \triangleright (r \mapsto 3)$
5. $\text{「False」} * (r \mapsto 3) \triangleright (s \mapsto 4) * (r \mapsto 4)$
6. $(r \mapsto 4) * (s \mapsto 3) \triangleright \text{「False」}$
7. $(r \mapsto 4) * (r \mapsto 3) \triangleright \text{「False」}$
8. $(r \mapsto 3) * (r \mapsto 3) \triangleright \text{「False」}$

Heap implications: true or false?

1. $(r \mapsto 3) * (s \mapsto 4) \triangleright (s \mapsto 4) * (r \mapsto 3)$ true
2. $(r \mapsto 3) \triangleright (s \mapsto 4) * (r \mapsto 3)$ false
3. $(s \mapsto 4) * (r \mapsto 3) \triangleright (r \mapsto 4)$ false
4. $(s \mapsto 4) * (r \mapsto 3) \triangleright (r \mapsto 3)$
5. $\text{「False」} * (r \mapsto 3) \triangleright (s \mapsto 4) * (r \mapsto 4)$
6. $(r \mapsto 4) * (s \mapsto 3) \triangleright \text{「False」}$
7. $(r \mapsto 4) * (r \mapsto 3) \triangleright \text{「False」}$
8. $(r \mapsto 3) * (r \mapsto 3) \triangleright \text{「False」}$

Heap implications: true or false?

1. $(r \mapsto 3) * (s \mapsto 4) \triangleright (s \mapsto 4) * (r \mapsto 3)$ true
2. $(r \mapsto 3) \triangleright (s \mapsto 4) * (r \mapsto 3)$ false
3. $(s \mapsto 4) * (r \mapsto 3) \triangleright (r \mapsto 4)$ false
4. $(s \mapsto 4) * (r \mapsto 3) \triangleright (r \mapsto 3)$ false
5. $\text{「False」} * (r \mapsto 3) \triangleright (s \mapsto 4) * (r \mapsto 4)$
6. $(r \mapsto 4) * (s \mapsto 3) \triangleright \text{「False」}$
7. $(r \mapsto 4) * (r \mapsto 3) \triangleright \text{「False」}$
8. $(r \mapsto 3) * (r \mapsto 3) \triangleright \text{「False」}$

Heap implications: true or false?

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

Heap implications: true or false?

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

Heap implications: true or false?

1. $(r \mapsto 3) * (s \mapsto 4) \triangleright (s \mapsto 4) * (r \mapsto 3)$ true
2. $(r \mapsto 3) \triangleright (s \mapsto 4) * (r \mapsto 3)$ false
3. $(s \mapsto 4) * (r \mapsto 3) \triangleright (r \mapsto 4)$ false
4. $(s \mapsto 4) * (r \mapsto 3) \triangleright (r \mapsto 3)$ false
5. $\text{「False」} * (r \mapsto 3) \triangleright (s \mapsto 4) * (r \mapsto 4)$ true
6. $(r \mapsto 4) * (s \mapsto 3) \triangleright \text{「False」}$ false
7. $(r \mapsto 4) * (r \mapsto 3) \triangleright \text{「False」}$ true
8. $(r \mapsto 3) * (r \mapsto 3) \triangleright \text{「False」}$

Heap implications: true or false?

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

Instantiation of existentials and propositions

$$(r \mapsto 6) \triangleright (\exists n. (r \mapsto n) * \text{'even } n\text{'})$$

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

Rules:

$$\frac{H_1 \triangleright ([x \rightarrow v] H_2)}{H_1 \triangleright (\exists x. H_2)} \text{ EXISTS-R}$$

$$\frac{(H_1 \triangleright H_2) \quad P}{H_1 \triangleright (H_2 * \text{'P'})} \text{ PROP-R}$$

Instantiation of existentials and propositions

$$(r \mapsto 6) \triangleright (\exists n. (r \mapsto n) * \text{「even } n\text{」})$$

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

Rules:

$$\frac{H_1 \triangleright ([x \rightarrow v] H_2)}{H_1 \triangleright (\exists x. H_2)} \text{ EXISTS-R} \qquad \frac{(H_1 \triangleright H_2) \quad P}{H_1 \triangleright (H_2 * \text{「}P\text{」})} \text{ PROP-R}$$

Example:

$$\frac{\frac{\frac{\overline{(r \mapsto 6) \triangleright (r \mapsto 6)}}{\text{REFL}} \quad \frac{\overline{\text{even } 6}}{\text{MATH}}}{(r \mapsto 6) \triangleright (r \mapsto 6) * \text{「even } 6\text{」}} \text{ PROP-R}}{(r \mapsto 6) \triangleright [n \rightarrow 6] ((r \mapsto n) * \text{「even } n\text{」})} \text{ SUBST}}{(r \mapsto 6) \triangleright \exists n. (r \mapsto n) * \text{「even } n\text{」}} \text{ EXISTS-R}$$

Extraction of existentials and propositions

$$(\exists n. \text{'even } n\text{'} * (r \mapsto n)) \triangleright (\exists m. \text{'even } m\text{'} * (r \mapsto m + 2))$$

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

$$(r \mapsto n) \triangleright \exists m. \text{'even } m\text{'} * (r \mapsto m + 2)$$

Extraction of existentials and propositions

$$(\exists n. \text{'even } n \text{'} * (r \mapsto n)) \triangleright (\exists m. \text{'even } m \text{'} * (r \mapsto m + 2))$$

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

$$(r \mapsto n) \triangleright \exists m. \text{'even } m \text{'} * (r \mapsto m + 2)$$

Reasoning rules:

$$\frac{\forall x. (H_1 \triangleright H_2)}{(\exists x. H_1) \triangleright H_2} \text{ EXISTS-L}$$

$$\frac{P \Rightarrow (H_1 \triangleright H_2)}{(\text{'}P \text{'} * H_1) \triangleright H_2} \text{ PROP-L}$$

Extraction of existentials and propositions

$$(\exists n. \ulcorner \text{even } n \urcorner * (r \mapsto n)) \triangleright (\exists m. \ulcorner \text{even } m \urcorner * (r \mapsto m + 2))$$

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

$$(r \mapsto n) \triangleright \exists m. \ulcorner \text{even } m \urcorner * (r \mapsto m + 2)$$

Reasoning rules:

$$\frac{\forall x. (H_1 \triangleright H_2)}{(\exists x. H_1) \triangleright H_2} \text{ EXISTS-L}$$

$$\frac{P \Rightarrow (H_1 \triangleright H_2)}{(\ulcorner P \urcorner * H_1) \triangleright H_2} \text{ PROP-L}$$

Same with explicit proof contexts:

$$\frac{\Gamma, x : A \vdash H_1 \triangleright H_2}{\Gamma \vdash (\exists(x : A). H_1) \triangleright H_2} (x \notin H_2)$$

$$\frac{\Gamma, h : P \vdash H_1 \triangleright H_2}{\Gamma \vdash (\ulcorner P \urcorner * H_1) \triangleright H_2}$$

Heap implications: true or false?

1. $(r \mapsto 3) \triangleright \exists n. (r \mapsto n)$
2. $\exists n. (r \mapsto n) \triangleright (r \mapsto 3)$
3. $\exists n. (r \mapsto n) * \lceil n > 0 \rceil \triangleright \exists n. \lceil n > 1 \rceil * (r \mapsto (n - 1))$
4. $(r \mapsto 3) * (s \mapsto 3) \triangleright \exists n. (r \mapsto n) * (s \mapsto n)$
5. $\exists n. (r \mapsto n) * \lceil n > 0 \rceil * \lceil n < 0 \rceil \triangleright (r \mapsto m) * (r \mapsto m)$

Heap implications: true or false?

1. $(r \mapsto 3) \triangleright \exists n. (r \mapsto n)$ true
2. $\exists n. (r \mapsto n) \triangleright (r \mapsto 3)$
3. $\exists n. (r \mapsto n) * \lceil n > 0 \rceil \triangleright \exists n. \lceil n > 1 \rceil * (r \mapsto (n - 1))$
4. $(r \mapsto 3) * (s \mapsto 3) \triangleright \exists n. (r \mapsto n) * (s \mapsto n)$
5. $\exists n. (r \mapsto n) * \lceil n > 0 \rceil * \lceil n < 0 \rceil \triangleright (r \mapsto m) * (r \mapsto m)$

Heap implications: true or false?

1. $(r \mapsto 3) \triangleright \exists n. (r \mapsto n)$ true
2. $\exists n. (r \mapsto n) \triangleright (r \mapsto 3)$ false
3. $\exists n. (r \mapsto n) * \lceil n > 0 \rceil \triangleright \exists n. \lceil n > 1 \rceil * (r \mapsto (n - 1))$
4. $(r \mapsto 3) * (s \mapsto 3) \triangleright \exists n. (r \mapsto n) * (s \mapsto n)$
5. $\exists n. (r \mapsto n) * \lceil n > 0 \rceil * \lceil n < 0 \rceil \triangleright (r \mapsto m) * (r \mapsto m)$

Heap implications: true or false?

1. $(r \mapsto 3) \triangleright \exists n. (r \mapsto n)$ true
2. $\exists n. (r \mapsto n) \triangleright (r \mapsto 3)$ false
3. $\exists n. (r \mapsto n) * \lceil n > 0 \rceil \triangleright \exists n. \lceil n > 1 \rceil * (r \mapsto (n - 1))$ true
4. $(r \mapsto 3) * (s \mapsto 3) \triangleright \exists n. (r \mapsto n) * (s \mapsto n)$
5. $\exists n. (r \mapsto n) * \lceil n > 0 \rceil * \lceil n < 0 \rceil \triangleright (r \mapsto m) * (r \mapsto m)$

Heap implications: true or false?

1. $(r \mapsto 3) \triangleright \exists n. (r \mapsto n)$ true
2. $\exists n. (r \mapsto n) \triangleright (r \mapsto 3)$ false
3. $\exists n. (r \mapsto n) * \lceil n > 0 \rceil \triangleright \exists n. \lceil n > 1 \rceil * (r \mapsto (n - 1))$ true
4. $(r \mapsto 3) * (s \mapsto 3) \triangleright \exists n. (r \mapsto n) * (s \mapsto n)$ true
5. $\exists n. (r \mapsto n) * \lceil n > 0 \rceil * \lceil n < 0 \rceil \triangleright (r \mapsto m) * (r \mapsto m)$

Heap implications: true or false?

1. $(r \mapsto 3) \triangleright \exists n. (r \mapsto n)$ true
2. $\exists n. (r \mapsto n) \triangleright (r \mapsto 3)$ false
3. $\exists n. (r \mapsto n) * \lceil n > 0 \rceil \triangleright \exists n. \lceil n > 1 \rceil * (r \mapsto (n - 1))$ true
4. $(r \mapsto 3) * (s \mapsto 3) \triangleright \exists n. (r \mapsto n) * (s \mapsto n)$ true
5. $\exists n. (r \mapsto n) * \lceil n > 0 \rceil * \lceil n < 0 \rceil \triangleright (r \mapsto m) * (r \mapsto m)$ 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:

$$\frac{}{a : \text{int}, a > 5 \vdash (r \mapsto 3) * (s \mapsto a) \triangleright (r \mapsto 3) * (s \mapsto a)}$$
$$\frac{}{a : \text{int}, a > 5 \vdash (r \mapsto 3) * (s \mapsto a) \triangleright (r \mapsto 3) * (s \mapsto 3 + (a - 3))}$$
$$\frac{}{a : \text{int}, a > 5 \vdash (r \mapsto 3) * (s \mapsto a) \triangleright \exists m. (r \mapsto 3) * (s \mapsto 3 + m)}$$
$$\frac{}{a : \text{int}, a > 5 \vdash (r \mapsto 3) * (s \mapsto a) \triangleright \exists nm. (r \mapsto n) * (s \mapsto n + m)}$$
$$\frac{}{\emptyset \vdash \exists a. \lceil a > 5 \rceil * (r \mapsto 3) * (s \mapsto a) \triangleright \exists nm. (r \mapsto n) * (s \mapsto n + m)}$$
$$\frac{}{\emptyset \vdash (r \mapsto 3) * \exists a. \lceil a > 5 \rceil * (s \mapsto a) \triangleright \exists nm. (s \mapsto n + m) * (r \mapsto n)}$$

Summary

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

Existentials and pure facts may be extruded from stars.

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

“ $\top \triangleright H$ ” is always true.

“ $(r \mapsto n) * (r \mapsto m)$ ” is equivalent to “ \top ”.

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

$$\frac{H \triangleright H' \quad \{H'\} t \{Q'\} \quad Q' \triangleright Q}{\{H\} t \{Q\}} \text{CONSEQUENCE}$$

with the overloading:

$$Q' \triangleright Q \equiv \forall x. (Q' x \triangleright Q x)$$

Consequence rule

$$\frac{H \triangleright H' \quad \{H'\} t \{Q'\} \quad Q' \triangleright Q}{\{H\} t \{Q\}} \text{CONSEQUENCE}$$

with the overloading:

$$Q' \triangleright Q \equiv \forall x. (Q' x \triangleright Q x)$$

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:

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

To:

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

Recall the need for garbage collection

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

From:

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

To:

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

Can the following rule be used?

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

Garbage collection rule

$$\frac{\{H\} t \{Q \star \text{GC}\}}{\{H\} t \{Q\}} \text{GC-POST} \quad \text{where: } \text{GC} \equiv \exists H'. H'$$

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

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

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

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

Garbage collection in the pre-condition

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

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

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

Proof:

$$\frac{\frac{\{H\} t \{Q\}}{\{H * GC\} t \{Q * GC\}} \text{FRAME}}{\{H * GC\} t \{Q\}} \text{GC-POST}$$

Combined structural rule

$$\frac{H \triangleright H' \quad \{H'\} t \{Q'\} \quad Q' \triangleright Q}{\{H\} t \{Q\}} \text{ CONSEQUENCE}$$

$$\frac{\{H\} t \{Q \star \text{GC}\}}{\{H\} t \{Q\}} \text{ GC-POST}$$

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

Combined structural rule

$$\frac{H \triangleright H' \quad \{H'\} t \{Q'\} \quad Q' \triangleright Q}{\{H\} t \{Q\}} \text{ CONSEQUENCE}$$

$$\frac{\{H\} t \{Q \star \text{GC}\}}{\{H\} t \{Q\}} \text{ GC-POST}$$

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

$$\frac{H = H_1 \star H_2 \quad \{H_1\} t \{Q_1\} \quad Q_1 \star H_2 = Q}{\{H\} t \{Q\}} \text{ FRAME'}$$

Combined structural rule

$$\frac{H \triangleright H' \quad \{H'\} t \{Q'\} \quad Q' \triangleright Q}{\{H\} t \{Q\}} \text{ CONSEQUENCE}$$

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

$$\frac{H = H_1 \star H_2 \quad \{H_1\} t \{Q_1\} \quad Q_1 \star H_2 = Q}{\{H\} t \{Q\}} \text{ FRAME'}$$

$$\frac{H \triangleright H_1 \star H_2 \quad \{H_1\} t \{Q_1\} \quad Q_1 \star H_2 \triangleright Q \star \text{GC}}{\{H\} t \{Q\}} \text{ COMBINED}$$

Extraction of existentials and propositions

$$\{\exists n. (r \mapsto n) * \ulcorner \text{even } n \urcorner\} (!r) \{\lambda x. \dots\}$$

To prove the above, we need to show that:

$$\forall n. \text{even } n \Rightarrow \{r \mapsto n\} (!r) \{\lambda x. \dots\}$$

Rules:

$$\frac{\forall x. \{H\} t \{Q\}}{\{\exists x. H\} t \{Q\}} \text{ EXISTS}$$

$$\frac{P \Rightarrow \{H\} t \{Q\}}{\{\ulcorner P \urcorner * H\} t \{Q\}} \text{ PROP}$$

Application: copying a tree with invariants

Specification of copy for binary trees:

$$\{p \rightsquigarrow \text{Mtree } T\} (\text{copy } p) \{\lambda p'. p \rightsquigarrow \text{Mtree } T * p' \rightsquigarrow \text{Mtree } T\}$$

Description of complete binary trees:

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

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:

$$\{p \rightsquigarrow \text{Mtree } T\} (\text{copy } p) \{\lambda p'. p \rightsquigarrow \text{Mtree } T * p' \rightsquigarrow \text{Mtree } T\}$$

Description of complete binary trees:

$$p \rightsquigarrow \text{MtreeComplete } T \equiv \exists n. (p \rightsquigarrow \text{Mtree } T) * \text{「depth } n T\text{」}$$

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

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

Proof of the derived specification

(1) By unfolding of MtreeComplete:

$$\{\exists n. (p \rightsquigarrow \text{Mtree } T) * \ulcorner \text{depth } n T \urcorner\}$$

(copy p)

$$\{\lambda p'. \quad \exists n. (p \rightsquigarrow \text{Mtree } T) * \ulcorner \text{depth } n T \urcorner \quad \} \\ * \exists n. (p' \rightsquigarrow \text{Mtree } T) * \ulcorner \text{depth } n T \urcorner$$

Proof of the derived specification

(1) By unfolding of MtreeComplete:

$$\begin{aligned} & \{ \exists n. (p \rightsquigarrow \text{Mtree } T) * \text{depth } n T \} \\ & \text{(copy p)} \\ & \{ \lambda p'. \exists n. (p \rightsquigarrow \text{Mtree } T) * \text{depth } n T \} \\ & * \exists n. (p' \rightsquigarrow \text{Mtree } T) * \text{depth } n T \end{aligned}$$

(2) By the EXISTS and PROP rules:

$$\begin{aligned} \forall n. \text{depth } n T & \Rightarrow \{ p \rightsquigarrow \text{Mtree } T \} \\ & \text{(copy p)} \\ & \{ \lambda p'. \exists n. (p \rightsquigarrow \text{Mtree } T) * \text{depth } n T \} \\ & * \exists n. (p' \rightsquigarrow \text{Mtree } T) * \text{depth } n T \end{aligned}$$

Proof of the derived specification

(1) By unfolding of MtreeComplete:

$$\begin{aligned} & \{ \exists n. (p \rightsquigarrow \text{Mtree } T) * \ulcorner \text{depth } n T \urcorner \} \\ & \text{(copy p)} \\ & \{ \lambda p'. \quad \exists n. (p \rightsquigarrow \text{Mtree } T) * \ulcorner \text{depth } n T \urcorner \quad \} \\ & \quad * \exists n. (p' \rightsquigarrow \text{Mtree } T) * \ulcorner \text{depth } n T \urcorner \end{aligned}$$

(2) By the EXISTS and PROP rules:

$$\begin{aligned} \forall n. \text{depth } n T \Rightarrow \{ p \rightsquigarrow \text{Mtree } T \} \\ & \text{(copy p)} \\ & \{ \lambda p'. \quad \exists n. (p \rightsquigarrow \text{Mtree } T) * \ulcorner \text{depth } n T \urcorner \quad \} \\ & \quad * \exists n. (p' \rightsquigarrow \text{Mtree } T) * \ulcorner \text{depth } n T \urcorner \end{aligned}$$

(3) By the CONSEQUENCE rule:

$$p \rightsquigarrow \text{Mtree } T * p' \rightsquigarrow \text{Mtree } T \triangleright \begin{aligned} & \exists n. (p \rightsquigarrow \text{Mtree } T) * \ulcorner \text{depth } n T \urcorner \\ & * \exists n. (p' \rightsquigarrow \text{Mtree } T) * \ulcorner \text{depth } n T \urcorner \end{aligned}$$

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

Proof of the derived specification

Coq demo.

Summary

Structural rules:

$$\frac{H \triangleright H_1 * H_2 \quad \{H_1\} t \{Q_1\} \quad Q_1 \star H_2 \triangleright Q \star GC}{\{H\} t \{Q\}} \text{ COMBINED}$$

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

Other structural rules are derivable.

Chapter 11

Reasoning rules for terms

Reasoning rule for sequences

Example:

$$\frac{\{r \mapsto n\} (\text{incr } r) \{\lambda_. r \mapsto n + 1\} \quad \{r \mapsto n + 1\} (!r) \{\lambda x. \ulcorner x = n + 1 \urcorner * r \mapsto n + 1\}}{\{r \mapsto n\} (\text{incr } r; !r) \{\lambda x. \ulcorner x = n + 1 \urcorner * r \mapsto n + 1\}}$$

Reasoning rule for sequences

Example:

$$\frac{\{r \mapsto n\} (\text{incr } r) \{\lambda_. r \mapsto n + 1\} \quad \{r \mapsto n + 1\} (!r) \{\lambda x. \ulcorner x = n + 1 \urcorner * r \mapsto n + 1\}}{\{r \mapsto n\} (\text{incr } r; !r) \{\lambda x. \ulcorner x = n + 1 \urcorner * r \mapsto n + 1\}}$$

Exercise: complete the rule for sequences.

$$\frac{\{\dots\} t_1 \{\dots\} \quad \{\dots\} t_2 \{\dots\}}{\{H\} (t_1; t_2) \{Q\}}$$

Reasoning rule for sequences

Solution 1:

$$\frac{\{H\} t_1 \{\lambda_{\cdot}. H'\} \quad \{H'\} t_2 \{Q\}}{\{H\} (t_1 ; t_2) \{Q\}}$$

Solution 2:

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

Remark: $Q' = \lambda_{\cdot}. H'$ is equivalent to $Q' () = H'$.

Reasoning rule for let-bindings

Exercise: complete the reasoning rule for let-bindings.

$$\frac{\{\dots\} t_1 \{\dots\} \quad \forall x. (\{\dots\} t_2 \{\dots\})}{\{H\} (\text{let } x = t_1 \text{ in } t_2) \{Q\}}$$

Reasoning rule for let-bindings

Exercise: complete the reasoning rule for let-bindings.

$$\frac{\{\dots\} t_1 \{\dots\} \quad \forall x. (\{\dots\} t_2 \{\dots\})}{\{H\} (\text{let } x = t_1 \text{ in } t_2) \{Q\}}$$

Solution:

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

Example of let-binding

$$\frac{\{H\} t_1 \{Q'\} \quad \forall x. \{Q' x\} t_2 \{Q\}}{\{H\} (\text{let } x = t_1 \text{ in } t_2) \{Q\}}$$

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

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

Example of let-binding

$$\frac{\{H\} t_1 \{Q'\} \quad \forall x. \{Q' x\} t_2 \{Q\}}{\{H\} (\text{let } x = t_1 \text{ in } t_2) \{Q\}}$$

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

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

Solution:

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

Reasoning rule for values

Example:

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

Rule:

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

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

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

Reasoning rule for values

Example:

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

Rule:

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

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

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

Solution:

$$\frac{H \triangleright Q v}{\{H\} v \{Q\}} \text{VAL-FRAME}$$

Derivability of the val-frame rule

$$\frac{H \triangleright Q v}{\{H\} v \{Q\}} \text{ VAL-FRAME}$$

Proof:

$$\frac{\frac{\frac{\frac{\overline{\{ \ulcorner \urcorner \} v \{ \lambda x. \ulcorner x = v \urcorner \}} \text{ VAL}}{\{H\} v \{ \lambda x. \ulcorner x = v \urcorner * H \}} \text{ FRAME}}{\{H\} v \{ \lambda x. \ulcorner x = v \urcorner * H \}} \text{ FRAME}}{\frac{\frac{\frac{\overline{H \triangleright Q v} \text{ HYPOTHESIS}}{\forall x. x = v \Rightarrow (H \triangleright Q x)} \text{ SUBST}}{\forall x. (\ulcorner x = v \urcorner * H) \triangleright (Q x)} \text{ PROP-L}}{(\lambda x. \ulcorner x = v \urcorner * H) \triangleright Q} \text{ DEF OF } \triangleright}}{\{H\} v \{Q\}} \text{ CONSEQ}}$$

Reasoning rule for conditionals

Rule:

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

Reasoning rule for conditionals

Rule:

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

Transformation to A-normal form:

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

Reasoning rule for top-level functions

Rule:

$$\frac{v_1 = \lambda x. t \quad \{H\} ([x \rightarrow v_2] t) \{Q\}}{\{H\} (v_1 v_2) \{Q\}} \text{TOP-FUN}$$

Transformation to A-normal form:

$$(t_1 t_2) = (\text{let } f = t_1 \text{ in let } v = t_2 \text{ in } (f v))$$

Verification of a simple function

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

Specification:

$$\forall rn. \{r \mapsto n\} (\text{incr } r) \{\lambda_. r \mapsto n + 1\}$$

Verification:

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

$$\{r \mapsto n\} (\text{let } a = !r \text{ in } r := a+1) \{\lambda_. r \mapsto n + 1\}$$

Verification of a simple function

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

Specification:

$$\forall rn. \{r \mapsto n\} (\text{incr } r) \{\lambda_. r \mapsto n + 1\}$$

Verification:

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

$$\{r \mapsto n\} (\text{let } a = !r \text{ in } r := a+1) \{\lambda_. r \mapsto n + 1\}$$

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:

$$\frac{v_1 = \mu f. \lambda x. t \quad \{H\} ([f \rightarrow v_1] [x \rightarrow v_2] t) \{Q\}}{\{H\} (v_1 v_2) \{Q\}} \text{TOP-FIX}$$

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:

$$\forall pL. \{p \rightsquigarrow \text{MList } L\} (\text{mlength } p) \{\lambda n. \ulcorner n = |L| \urcorner * p \rightsquigarrow \text{MList } L\}$$

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:

$$\forall pL. \{p \rightsquigarrow \text{MList } L\} (\text{mlength } p) \{\lambda n. \ulcorner n = |L| \urcorner * p \rightsquigarrow \text{MList } L\}$$

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

Verification of mlength: nil case

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

$$\{p \rightsquigarrow \text{MList } L\} (0) \{\lambda n. \ulcorner n = |L| \urcorner * p \rightsquigarrow \text{MList } L\}$$

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

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

- Replace L with nil .

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

- Apply the VAL-FRAME rule.

$$\ulcorner \urcorner \triangleright \ulcorner 0 = 0 \urcorner * \ulcorner \text{nil} = \text{nil} \urcorner$$

Verification of mlength: cons case (1/2)

Case $p \neq \text{null}$. Goal is:

$$\begin{aligned} & \{p \rightsquigarrow \text{MList } L\} \\ & (\text{let } p' = p.\text{tl} \text{ in let } n' = \text{mlength } p' \text{ in } 1 + n') \\ & \{\lambda n. \ulcorner n = |L| \urcorner * p \rightsquigarrow \text{MList } L\} \end{aligned}$$

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

$$p' \rightsquigarrow \text{MList } L' * p \rightsquigarrow \{\text{hd}=x; \text{tl}=p'\}$$

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

$$\begin{aligned} & \{p' \rightsquigarrow \text{MList } L' * p \rightsquigarrow \{\text{hd}=x; \text{tl}=p'\}\} \\ & (\text{let } n' = \text{mlength } p' \text{ in } 1 + n') \\ & \{\lambda n. \ulcorner n = |L| \urcorner * p' \rightsquigarrow \text{MList } L' * p \rightsquigarrow \{\text{hd}=x; \text{tl}=p'\}\} \end{aligned}$$

– Apply the frame rule to remove: $p \rightsquigarrow \{\text{hd}=x; \text{tl}=p'\}$.

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

Verification of mlength: cons case (2/2)

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

– First branch, exploit the induction hypothesis:

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

– Second branch:

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

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

$$n' = |L'| \quad \Rightarrow \quad p' \rightsquigarrow \text{MList } L' \triangleright \ulcorner 1 + n' = |L| \urcorner * p' \rightsquigarrow \text{MList } L'$$

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

Reasoning rule for local functions

Rule template:

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

Hypothesis about f :

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

Rule:

$$\frac{\forall f. Pf \Rightarrow \{H\} t \{Q\} \quad Pf = (\forall x H' Q'. \{H'\} \text{body} \{Q'\} \Rightarrow \{H'\} (f x) \{Q'\})}{\{H\} (\text{let rec } f x = \text{body in } t) \{Q\}} \text{FIX}$$

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 $L = x :: L'$.

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

Summary of chapter 8

Small footprint specification for C-style memory accesses:

$$\begin{aligned} & \{p \mapsto w\} (*p = v) \quad \{\lambda_. p \mapsto v\} \\ & \{p \mapsto v\} (*p) \quad \{\lambda x. \ulcorner x = v \urcorner * p \mapsto v\} \end{aligned}$$

Representation of a full array using a list:

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

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

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

Summary of chapter 9

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

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

“ $\top \triangleright H$ ” is always true.

“ $(r \mapsto n) * (r \mapsto m)$ ” is equivalent to “ \top ”.

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

Summary of chapter 10

Structural rules:

$$\frac{H \triangleright H_1 * H_2 \quad \{H_1\} t \{Q_1\} \quad Q_1 \star H_2 \triangleright Q \star GC}{\{H\} t \{Q\}} \text{ COMBINED}$$

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

Other structural rules are derivable.

Exercises

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

Available from the webpage of the course.

The end!