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

Exercise: specify the tree copy function.

Exercise: describe the frame process for tree copy.

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

{ } (Array.get i p) { }
{ } (Array.set i p v){ }
{ } (Array.length p) { }

Exercise: give a small-footprint specification for quicksort.
For each of the heap implications below, state whether it is true or false.

1. \((r \mapsto 3) \times (s \mapsto 4) \models (s \mapsto 4) \times (r \mapsto 3)\)
2. \((r \mapsto 3) \models (s \mapsto 4) \times (r \mapsto 3)\)
3. \((s \mapsto 4) \times (r \mapsto 3) \models (r \mapsto 4)\)
4. \((s \mapsto 4) \times (r \mapsto 3) \models (r \mapsto 3)\)
5. \([\text{False}] \times (r \mapsto 3) \models (s \mapsto 4) \times (r \mapsto 4)\)
6. \((r \mapsto 4) \times (s \mapsto 3) \models [\text{False}]\)
7. \((r \mapsto 4) \times (r \mapsto 3) \models [\text{False}]\)
8. \((r \mapsto 3) \times (r \mapsto 3) \models [\text{False}]\)

For each of the heap implications below, state whether it is true or false.

1. \((r \mapsto 3) \models \exists n. (r \mapsto n)\)
2. \(\exists n. (r \mapsto n) \models (r \mapsto 3)\)
3. \(\exists n. (r \mapsto n) \times [n > 0] \models \exists n. [n > 1] \times (r \mapsto (n - 1))\)
4. \((r \mapsto 3) \times (s \mapsto 3) \models \exists n. (r \mapsto n) \times (s \mapsto n)\)
5. \(\exists n. (r \mapsto n) \times [n > 0] \times [n < 0] \models (r \mapsto n) \times (r \mapsto n)\)

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

\[
\begin{align*}
\{H\} & \quad t \quad \{Q\} \\
\hline
{H \ast \text{GC}} & \quad t \quad \{Q\}
\end{align*}
\]
**Exercise:** give a specification of `copy` in terms of `MtreeComplete`; which rules are used to derive this specification?

**Exercise:** complete the rule for sequences.

\[
\begin{array}{c}
\{ \quad \} t_1 \{ \quad \} \quad \{ \quad \} t_2 \{ \quad \} \\
\{H\} (t_1 ; t_2) \{Q\}
\end{array}
\]

**Exercise:** complete the reasoning rule for let-bindings.

\[
\begin{array}{c}
\{ \quad \} t_1 \{ \quad \} \quad \forall x. \{ \quad \} t_2 \{ \quad \} \\
\{H\} (\text{let } x = t_1 \text{ in } t_2) \{Q\}
\end{array}
\]

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

\[
\{r \rightarrow 3\} (\text{let } a = !r \text{ in } a+1) \{Q\}
\]

Solution:

\[
\begin{align*}
H & \equiv \\
Q & \equiv \\
Q' & \equiv
\end{align*}
\]

Reasoning rule for values:

\[
\begin{array}{c}
\Rightarrow \\
\{H\} v \{Q\}
\end{array}
\]
**Exercise:** specify the interface for mutable queues in terms of:

\[ p \leadsto \text{Queue } L \]

Remark: items are pushed to the back of list, and popped from the head; transfer migrates items from the second queue to the back of the first.

\[
(\text{create }())
\]

\[
(\text{is\_empty } q)
\]

\[
(\text{push } x \ q)
\]

\[
(\text{pop } q)
\]

\[
(\text{transfer } q1 \ q2)
\]