Interactive Software Verification

Spring Term 2013

Holger Gast

gasth@in.tum.de

11.6.2013

Today

- Techniques
  - Abstractions predicates/functions
  - Separation lemmas
- Examples
  - Filling an array
  - Partition re-verified
  - Bubble sort

Remark: Semantics of Short-circuit Ops

- Short-cirucit semantics of `&&` and `||`:
  Execute right-hand side only if lhs does not determine result
- Example: protect array access

```
if (0<=i && i<n && a[i]>0) ...
```

- Embedding

IF 0 ≤ ´i
  THEN IF ´i < ´n
    THEN Guard "array-bounds" ⦃acheck ´i ´a⦄
      (IF 0 < aget ´a ´i THEN ´i :== 0 FI)
    FI
  FI

- Note: bounds-check protected by outer `IF`

⇒ Checked conditions available in proof

Remark: Semantics of Short-circuit Ops

- Proof obligation in example

  $\bigwedge a\, i\, n.\ [\![0 \leq i; i < \mathrm{alength}\ a]\!] \Longrightarrow \mathrm{acheck}\ i\ a$

- Optimization: inline if no guards occur

  IF $0 \leq$ ´i $\wedge$ ´i $<$ ´n
  THEN Guard "array-bounds" $\{\!|$acheck ´i ´a$|\!\}$
  (IF $0 <$ aget ´a ´i THEN ´i :== 0 FI)
  FI

Recap: Strategies for Loop Invariants

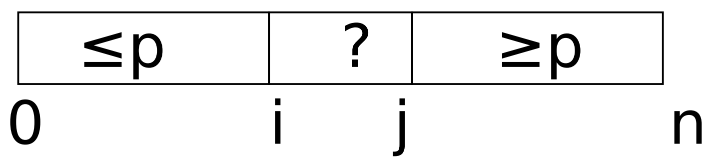
- Generalize the desired post-condition
  - Loop test is $a \neq b$, then post-condition from $a = b \wedge I$
  - If loop test is $a < b$, then add $a \leq b$ to invariant
- Describe (precisely) the achieved partial result
  - Consider the variables that are modified in the body
  - And capture the relationship between them
  - . . . and the relationship to the input values (logical vars)
- Add safety assertions
  - About the range of index variables
  - About pointers not being `null`
- Preserve information from before loop

Example: Partition

- Desired post-condition

  $(\forall\, k \in \{\, 0 \,..{<}\, i \,\}.\ \mathrm{aget\,a\,k} \leq p) \wedge$
  $(\forall\, k \in \{\, i \,..{<}\, \mathrm{alength\,a} \,\}.\ \mathrm{aget\,a\,k} > p)$

- Idea



- Invariant—main part

  $(\forall\, k \in \{\, 0 \,..{<}\, i \,\}.\ \mathrm{aget\,a\,k} \leq p) \wedge$
  $(\forall\, k \in \{\, j \,..{<}\, \mathrm{alength\,a} \,\}.\ \mathrm{aget\,a\,k} > p) \wedge$

- Invariant—safety assertions

  $0 \leq i \wedge i \leq j \wedge j \leq \mathrm{alength\,a} \wedge$

- Invariant—preserved information

  $n = \mathrm{alength\,a}$

Abstraction Predicates

Arrays So Far

- Statements about arrays so far low-level

- Example: loop invariant of partition

  $$(\forall\, \mathsf{k} \in \{\, \mathsf{0} \,..\!<\, \mathsf{i} \,\}.\ \mathsf{aget\,a\,k} \leq \mathsf{p}) \wedge$$
  $$(\forall\, \mathsf{k} \in \{\, \mathsf{j} \,..\!<\, \mathsf{alength\,a} \,\}.\ \mathsf{aget\,a\,k} > \mathsf{p}) \wedge \ldots$$

- View so far: arrays as—well—arrays

- . . . but that's not what we usually like to think about

Abstraction & Programming

- Crucial to programming: encapsulation & abstraction

- Library for hash-maps
  - Internally use arrays & buckets for collisions
  - Externally: pretend to represent `int` $\rightharpoonup$ `int`

⇒ Desirable: abstraction predicate

  hash-map a m

  - a is the array
  - m is the abstract represented map value

- Today: work out the issues with basic examples

Abstraction on Arrays

- Usual idea: array as representation for sequence
  - Indexing not really adequate
  - Talk about "sequence stored in array" instead
  - Example: $\mathsf{sorted}\,(\mathsf{elems}\,\mathsf{a}\,0\,\mathsf{N})$

⇒ Define function $\mathsf{elems}\,\mathsf{a}\,\mathsf{i}\,\mathsf{j}$
  - Yield `'a` sequence from `'a` array
  - Access slice $[i, j)$

- Definition (for those who are curious/functionally inclined)

$$\mathsf{elems}\,\mathsf{a}\,\mathsf{i}\,\mathsf{j} \equiv \mathsf{map}\,(\mathsf{aget}\,\mathsf{a} \circ \mathsf{int})\,[\mathsf{nat}\,\mathsf{i}\,..{<}\,\mathsf{nat}\,\mathsf{j}]$$

  - Re-use notion & theory of `upto`
  - Use array as a mapping from indizes to values
  - Cast from `int` to `nat` where necessary

Verifying Array Initialization

- First: version with explicit indices

```
i = 0;
/*@ 0 ≤ i ∧ i ≤ n ∧ n ≤ alength a ∧ n = N ∧
  (∀ j ∈ {0..<i}. aget a j = k)
*/
while (i != n) {
 a[i] = k;
 i = i + 1;
}
```

- Post-condition

$$\forall\, j \in \{0..{<}N\}.\ \mathrm{aget}\ a\ j = k$$

- Proof: straightforward (because Isabelle knows about sets)

Creating a Sequence

- Read/use array initialization as sequence creation

- Abstracted post-condition

  $\mathsf{elems\,a\,0\,N = replicate\,K\,N}$

- With previous loop-invariant: prove in the end

  $[\![0 \le \mathsf{n};\, \mathsf{n} \le \mathsf{alength\,a};\, \forall\, \mathsf{j} \in \{0..<\mathsf{n}\}.\ \mathsf{aget\,a\,j} = \mathsf{k}$
  $]\!] \Longrightarrow \mathsf{elems\,a\,0\,n = replicate\,(nat\,n)\,k}$

- Requires some thought & lemma `map_replicate_const`:

  $\mathsf{map}\,(\lambda \mathsf{x}.\ \mathsf{k})\,\mathsf{xs} = \mathsf{replicate}\,(\mathsf{length\,xs})\,\mathsf{k}$

Re-phrasing the Invariant

- Use standard strategy: invariant is partial result towards desired post-condition

⇒ Yields loop invariant

$$0 \le i \wedge i \le n \wedge n \le \mathrm{alength}\, a \wedge n = N \wedge k = K \wedge$$
$$\mathbf{elems\, a\, 0\, i = replicate\, (nat\, i)\, k}$$

⇒ Proof of post-condition trivial

- Preserving loop invariant yields goal

$$\begin{array}{l} [\![0 \le i; i \le n; n \le \mathrm{alength}\, a; \\ \quad \mathrm{elems}\, a\, 0\, i = \mathrm{replicate}\, (\mathrm{nat}\, i)\, k; \\ \quad i \ne n \\ ]\!] \Longrightarrow \mathrm{elems}\, (\mathrm{aset}\, i\, k\, a)\, 0\, (i+1) = \mathrm{replicate}\, (\mathrm{nat}\, (i+1))\, k \end{array}$$

Issues in Proving the Invariant

$$\llbracket 0 \le i; i \le n; n \le \mathsf{alength}\ a;$$
$$\quad \mathsf{elems}\ a\ 0\ i = \mathsf{replicate}\ (\mathsf{nat}\ i)\ k;$$
$$\quad i \ne n$$
$$\rrbracket \Longrightarrow \mathsf{elems}\ (\mathsf{aset}\ i\ k\ a)\ 0\ (i+1) = \mathsf{replicate}\ (\mathsf{nat}\ (i+1))\ k$$

- Interaction: abstraction&modification

  $\ldots\ \mathsf{elems}\ (\mathsf{aset}\ i\ k\ a)\ 0\ (i+1)\ \ldots$

- Using the premises / old invariant / induction hypothesis

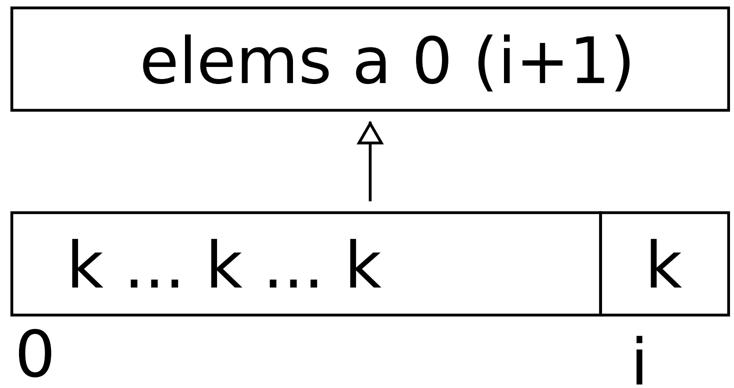
  $\mathsf{elems}\ a\ 0\ i = \mathsf{replicate}\ (\mathsf{nat}\ i)\ k;$
  $\ldots \Longrightarrow \mathsf{elems}\ (\mathsf{aset}\ i\ k\ a)\ 0\ (i+1) = \mathsf{replicate}\ (\mathsf{nat}\ (i+1))\ k$

- Reasoning within the abstraction

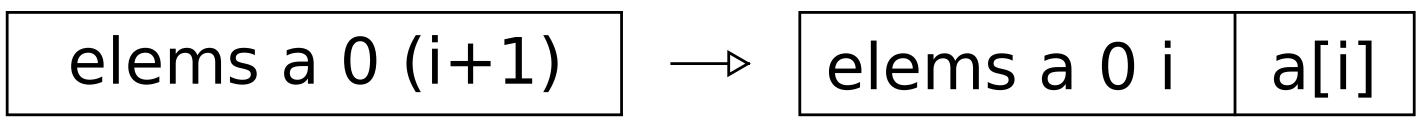
  $\mathsf{replicate}\ (\mathsf{nat}\ (i+1))\ k\ \mathsf{pd} = \ldots\ \mathsf{replicate}\ i\ k\ \ldots$

Idea: Splitting the Abstraction

- Idea of the loop step



⇒ Need to split the abstraction



- Achieved two objectives at once
  - Reasoning about place i seperately
  - Use loop invariant / induction hypothesis

Splitting `elems`

- General lemma: split elems at intermediate point k

  $$\llbracket 0 \leq i; i \leq k; k \leq j \rrbracket \Longrightarrow \mathsf{elems}\ a\ i\ j = \mathsf{elems}\ a\ i\ k\ @\ \mathsf{elems}\ a\ k\ j$$

- Special cases: first & last element

  $$\llbracket 0 \leq i; i < j \rrbracket \Longrightarrow \mathsf{elems}\ a\ i\ j = \mathbf{aget\ a\ i} \# \mathsf{elems}\ a\ (i+1)\ j$$
  $$\llbracket 0 \leq i; i < j \rrbracket \Longrightarrow \mathsf{elems}\ a\ i\ j = \mathsf{elems}\ a\ i\ (j-1)\ @\ [\mathbf{aget\ a\ (j-1)}]$$

- Idea: programs usually modify sequence at beginning/end

- Heuristics: split at modification point (similar to case distinctions in `partition`)

  $$\llbracket 0 \leq i; i \leq k; k < j \rrbracket \Longrightarrow \mathsf{elems}\ \mathbf{(aset\ k\ x\ a)}\ i\ j = \mathsf{elems}\ a\ i\ k\ @\ [x]\ @\ \mathsf{elems}\ a\ (k+1)\ j$$

Note: we know that at `k`, the value is `x`

Simplifying `elems`

- Anticipate special cases occurring in verification

- Empty sequence of elements before/end loop

  $\mathsf{elems}\ a\ i\ i = []$

- Singleton sequences (split off from beginning/end)

  $[\![\, 0 \le i \,]\!] \Longrightarrow \mathsf{elems}\ a\ i\ (i + 1) = [\mathsf{aget}\ a\ i]$
  $[\![\, 0 < i \,]\!] \Longrightarrow \mathsf{elems}\ a\ (i - 1)\ i = [\mathsf{aget}\ a\ (i - 1)]$

Proceeding with the Proof

- Original

  $\mathrm{elems}\,(\mathrm{aset}\;i\;k\;a)\;0\;(i+1) = \mathrm{replicate}\,(\mathrm{nat}\,(i+1))\;k$

- With `elems-split-last`

  $\mathrm{elems}\,(\mathrm{aset}\;i\;k\;a)\;0\;i\;@\;\mathbf{[k]} = \mathrm{replicate}\,(\mathrm{nat}\,(i+1))\;k$

- Splitting the replicate to align (`nat-add-distrib+replicate_append_same`)

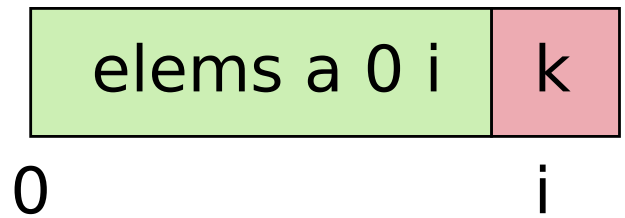
  $\mathrm{elems}\,(\mathrm{aset}\;i\;k\;a)\;0\;i\;@\;[k] = \mathrm{replicate}\;\mathbf{(nat\;i)}\;k\;@\;\mathbf{[k]}$

- Remains only

  $\mathrm{elems}\,(\mathrm{aset}\;i\;k\;a)\;0\;i = \mathrm{replicate}\,(\mathrm{nat}\;i)\;k$

Separation Lemmas

- Problematic: elems (aset i k a) 0 i = replicate (nat i) k



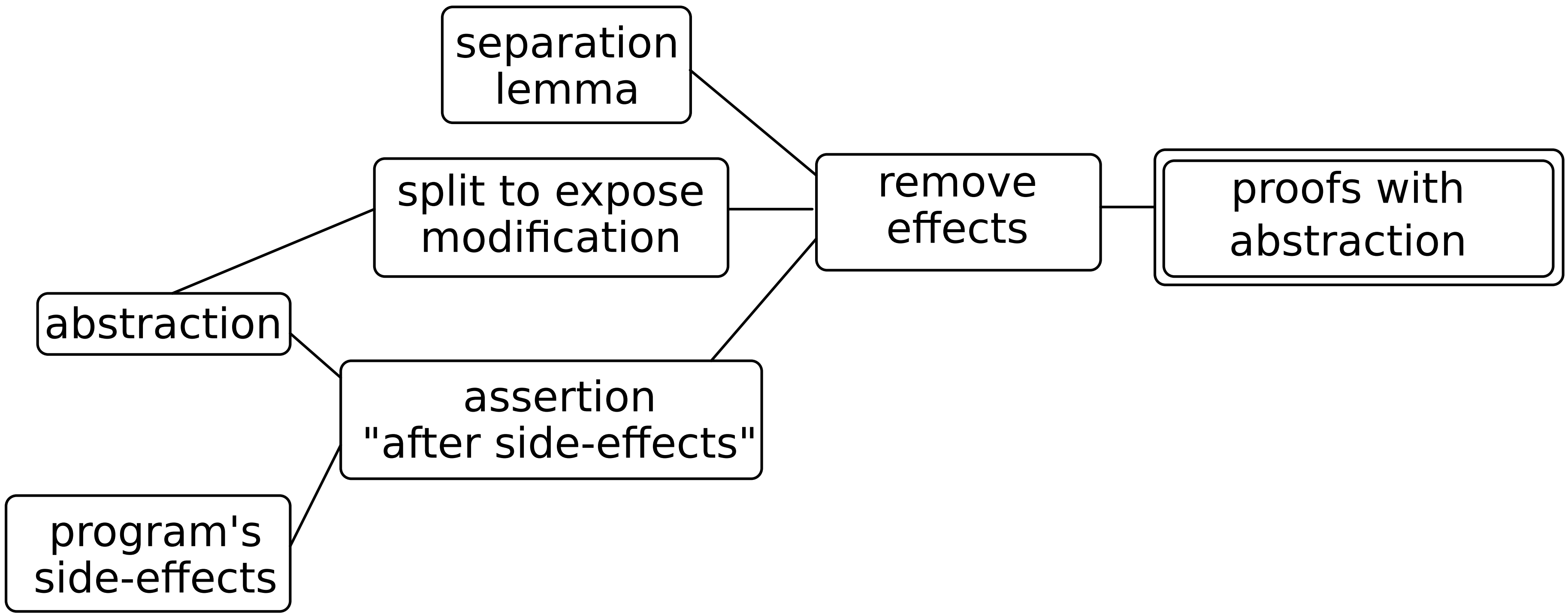
- Noted before: interaction read/update in proof obligations
- Had lemma to disregard irrelevant update

  $i \neq j \Longrightarrow \text{aget}\,(\text{aset}\; j\; x\; a)\; i = \text{aget}\; a\; i$

- Need similar lemmas for any introduced abstraction
- General idea: separation lemmas [1, 2]
- For the case of elems

  $k \notin \{i..<j\} \Longrightarrow \text{elems}\,(\text{aset}\; k\; y\; a)\; i\; j = \text{elems}\; a\; i\; j$

Working with Abstractions



Re-Verifying Partition

- Recall old formulation

≤p	?	≥p

0 i j n

$$(\forall\, k \in \{\, 0\, ..{<}\, i\, \}.\ \mathsf{aget}\ a\ k \leq p) \wedge$$
$$(\forall\, k \in \{\, j\, ..{<}\ \mathsf{alength}\ a\, \}.\ \mathsf{aget}\ a\ k > p)$$

- Proof was rather cumbersome, with case-distinctions

apply auto
apply (simp-all only: not-le not-less)
apply (case-tac "k = j - 1")
apply simp
apply simp
apply (case-tac "k < i")
apply simp
apply simp
. . . *three more case distinctions*. . .

Partition with Abstraction

- Re-phrase invariant (plus safety of indexes, as usual)

$$(\forall x \in \mathsf{set}\,(\mathsf{elems}\,a\,0\,i).\,x \leq p) \wedge$$
$$(\forall x \in \mathsf{set}\,(\mathsf{elems}\,a\,j\,(\mathsf{alength}\,a)).\,x > p)$$

- Proof-obligations have the form

$$\begin{array}{l}[\![\forall x{\in}\mathsf{set}\,(\mathsf{elems}\,a\,0\,i).\,x \leq p; \ldots\\ \quad \mathsf{aget}\,a\,i \leq p;\\ \quad x \in \mathsf{set}\,(\mathsf{elems}\,a\,0\,(i+1))\\ ]\!] \Longrightarrow x \leq p\end{array}$$

⇒ Splitting off the last element does the trick (use `fast`)

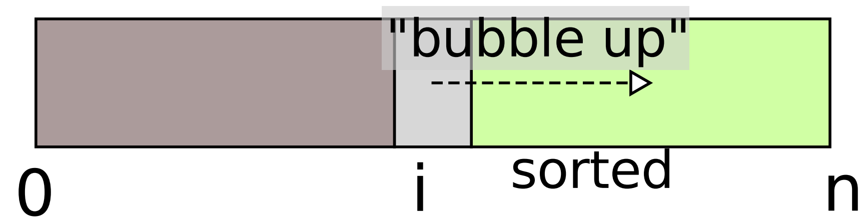
$$\begin{array}{l}[\![\forall x{\in}\mathsf{set}\,(\mathsf{elems}\,a\,0\,i).\,x \leq p; \ldots\\ \quad \mathsf{aget}\,a\,i \leq p;\\ \quad x = \mathsf{aget}\,a\,i \vee x \in \mathsf{set}\,(\mathsf{elems}\,a\,0\,i)\\ ]\!] \Longrightarrow x \leq p\end{array}$$

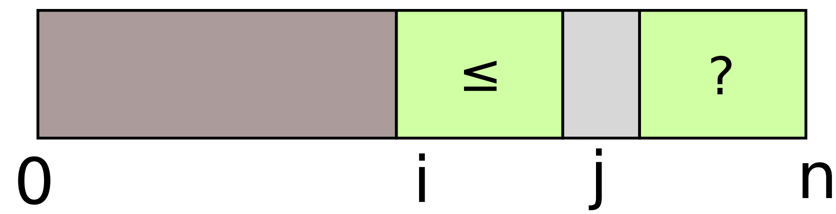
⇒ Introduces case-distinction along abstraction & program

Verifying Bubble-Sort

Bubble-Sort

```
i = n;
while (i != 0) {
 i = i - 1;
 j = i;
 while (j < n - 1 &&
          a[j + 1] < a[j]) {
   t = a[j];
   a[j] = a[j+1];
   a[j+1] = t;
   j = j + 1;
 }
}
```





Specification

- Pre-condition as usual: $\mathsf{n = alength\,a \wedge n = N \wedge 0 \leq n}$
- Invariant of outer loop

  $\mathsf{0 \leq i \wedge i \leq n \wedge n = N \wedge n = alength\,a \wedge}$
  $\mathsf{sorted\,(elems\,a\,i\,N)}$

- Invariant of inner loop (imagine that without elems & explicit $\forall$!

  $\mathsf{0 \leq i \wedge i \leq j \wedge j < n \wedge n = N \wedge n = alength\,a \wedge}$
  $\mathsf{sorted\,(elems\,a\,i\,j\,@\,elems\,a\,(j+1)\,n) \wedge}$
  $\mathsf{(\forall\, x \in set\,(elems\,a\,i\,j).\, x \leq aget\,a\,j)}$

- Note analogous "splitting" of sorted list by `sorted_append`

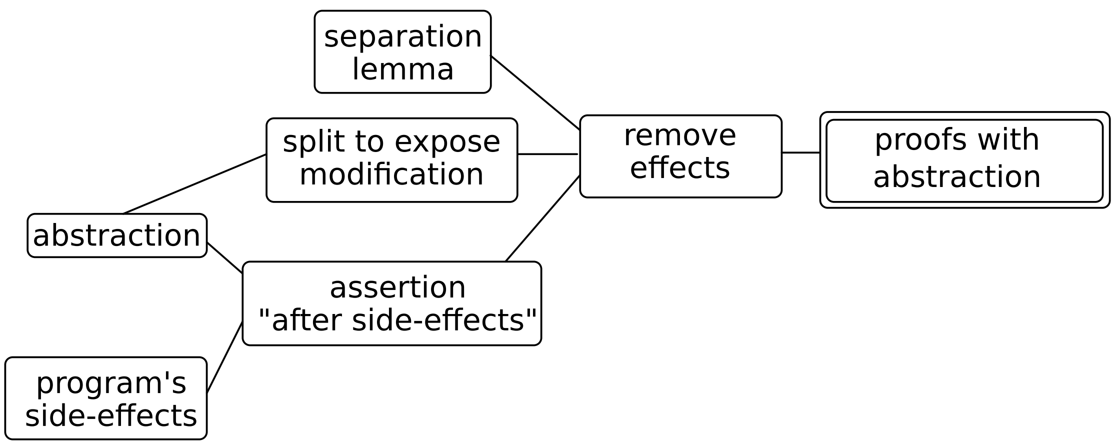
  $\mathsf{sorted\,(xs@ys) = (sorted\,xs\,\&\,sorted\,ys\,\&\,(\forall\, x \in set\,xs.\,\forall\, y \in set\,ys.\, x \leq y))}$

Proofs for Bubble-Sort

Interactively during the lecture, see `Demo08.thy`

Today

- Strategies for invariants by more examples
- Abstraction predicates/functions
- Separation lemmas



References

[1] Richard Bornat. Proving pointer programs in Hoare logic. In *Mathematics of Program Construction*, 2000.

[2] Farhad Mehta and Tobias Nipkow. Proving pointer programs in higher-order logic. *Inf. Comput.*, 199(1–2):200–227, 2005.