Towards Mechanized Program Verification with Separation Logic

Tjark Weber

webertj@in.tum.de

Technische Universität München

CSL, September 21, 2004

Motivation

- Separation logic: a program logic for pointer programs (Peter O'Hearn, John Reynolds et al.)
- Formal verification needs tool support

Motivation

- Separation logic: a program logic for pointer programs

  (Peter O'Hearn, John Reynolds et al.)

- Formal verification needs tool support

$\Rightarrow$ integration of separation logic with Isabelle/HOL

Overview

- The language
- Semantics
- Hoare logics
- The frame rule
- In-place list reversal

Stores, Heaps, States

types

$$
\begin{aligned}
\text{addr} &= \text{nat} \\
\text{val} &= \text{nat} \\
\text{store} &= \text{var} \Rightarrow \text{val} \\
\text{heap} &= \text{addr} \rightharpoonup \text{val} \\
\text{state} &= (\text{store} \times \text{heap})\ \text{option} \\
\text{aexp} &= \text{store} \Rightarrow \text{val} \\
\text{bexp} &= \text{store} \Rightarrow \text{bool}
\end{aligned}
$$

The Language: IMP . . .

- skip
- *var* $:==$ *aexp*
- *c1*; *c2*
- if *bexp* then *c1* else *c2*
- while *bexp* do *c1*

. . . with Pointers

- *var* $:==$ list *aexps* — allocation (records)
- *var* $:==$ alloc *aexp* — allocation (arrays)
- *var* $:==$ @*aexp* — lookup
- @*aexp1* $:==$ *aexp2* — mutation
- dispose *aexp* — deallocation

Disjoint Heaps, Union of Heaps

- Disjoint ($\bowtie$):

$$f \bowtie g \equiv \mathit{dom}\ f \cap \mathit{dom}\ g = \{\}$$

- Union (++):

$$f{+}{+}g \equiv \lambda x.\ \mathit{case}\ g\ x\ \mathit{of}\ \mathit{None} \Rightarrow f\ x \mid \mathit{Some}\ y \Rightarrow \mathit{Some}\ y$$

Disjoint Heaps, Union of Heaps

- Disjoint ($\bowtie$):

$$f \bowtie g \equiv \textit{dom}\ f \cap \textit{dom}\ g = \{\}$$

- Union (++):

$$f{+}{+}g \equiv \lambda x.\ \textit{case}\ g\ x\ \textit{of}\ \textit{None} \Rightarrow f\ x \mid \textit{Some}\ y \Rightarrow \textit{Some}\ y$$

- Taking the union of disjoint heaps is commutative:

$$f \bowtie g \Longrightarrow f \mathbin{++} g = g \mathbin{++} f$$

Operational Semantics: Allocation

- $[\![\textit{heap-isfree } h\ a\ (\textit{length as}); \textit{vs} = \textit{map}\ (\lambda e.\ e\ s)\ \textit{as}]\!]$
  $\Longrightarrow \langle x :== \text{list } \textit{as}, \textit{Some}\ (s, h) \rangle$
  $\longrightarrow_c \textit{Some}\ (s[x \mapsto a], \textit{heap-update } h\ a\ \textit{vs})$

- $\forall\, a.\ \neg\ \textit{heap-isfree } h\ a\ (\textit{length as}) \Longrightarrow$
  $\langle x :== \text{list } \textit{as}, \textit{Some}\ (s, h) \rangle \longrightarrow_c \textit{None}$

Operational Semantics: Lookup

- $a\ s \in dom\ h \Longrightarrow$
  $\langle x :== @a, Some\ (s, h)\rangle$
  $\longrightarrow_c Some\ (s[x \mapsto heap\text{-}lookup\ h\ (a\ s)], h)$

- $a\ s \notin dom\ h \Longrightarrow \langle x :== @a, Some\ (s, h)\rangle \longrightarrow_c None$

Operational Semantics: Mutation

- $\mathit{a\ s} \in \mathit{dom\ h} \Longrightarrow$
  $\langle @\mathit{a} :== \mathit{v}, \mathit{Some}\ (\mathit{s}, \mathit{h})\rangle$
  $\longrightarrow_c \mathit{Some}\ (\mathit{s}, \mathit{heap\text{-}update\ h}\ (\mathit{a\ s})\ [\mathit{v\ s}])$

- $\mathit{a\ s} \notin \mathit{dom\ h} \Longrightarrow \langle @\mathit{a} :== \mathit{v}, \mathit{Some}\ (\mathit{s}, \mathit{h})\rangle \longrightarrow_c \mathit{None}$

Operational Semantics: Deallocation

- $a\ s \in dom\ h \Longrightarrow$
  $\langle \text{dispose } a, Some\ (s, h)\rangle \longrightarrow_c Some\ (s, heap\text{-}remove\ h\ (a\ s))$

- $a\ s \notin dom\ h \Longrightarrow \langle \text{dispose } a, Some\ (s, h)\rangle \longrightarrow_c None$

Denotational Semantics

- Lookup:

$$
\begin{aligned}
&C\ (x :== @a) = \\
&\{(Some\ (s, h), \\
&\quad Some\ (s[x \mapsto heap\text{-}lookup\ h\ (a\ s)], h)) \mid \\
&s\ h.\ a\ s \in dom\ h\} \cup \\
&\{(Some\ (s, h), None) \mid s\ h.\ a\ s \notin dom\ h\} \cup \\
&\{(None, None)\}
\end{aligned}
$$

- Equivalence of denotational and operational semantics:

$$((s, t) \in C\ c) = \langle c,s\rangle \longrightarrow_c t$$

Separation Logic

- $\wedge$, $\vee$, $\neg$, $\longrightarrow$, ...

- Separating conjunction:

$$(P \wedge\!\! * \; Q)\; h \equiv \exists\, h'\, h''.\; h' \bowtie h'' \wedge h' +\!\!+\, h'' = h \wedge P\, h' \wedge Q\, h''$$

- Separating implication:

$$(P -\!\!* \; Q)\; h \equiv \forall\, h'.\; h' \bowtie h \wedge P\, h' \longrightarrow Q\; (h +\!\!+\, h')$$

Assertions

- $\mathit{emp}\ h \equiv \mathit{dom}\ h = \{\}$
- $(a \mapsto v)\ h \equiv \mathit{dom}\ h = \{a\} \wedge \mathit{heap\text{-}lookup}\ h\ a = v$
- $(a \mapsto -)\ h \equiv \exists\, v.\ (a \mapsto v)\ h$
- $a \hookrightarrow v \equiv a \mapsto v \wedge\!\!* \mathit{true}$

Some Properties of $\wedge *$

- $P \wedge * \left(Q \wedge * R\right) = P \wedge * Q \wedge * R$
- $P \wedge * Q = Q \wedge * P$
- $emp \wedge * P = P$
- $P \wedge * emp = P$
- $\ldots$

Hoare Logic: Partial Correctness

- $\models_p \{P\}\ c\ \{Q\} \equiv$
  $\forall s\ h\ s'\ h'.$
  $(Some\ (s, h), Some\ (s', h')) \in C\ c \longrightarrow P\ s\ h \longrightarrow Q\ s'\ h'$
  - Error state may be reachable
  - Partial correctness
- $\vdash_p \{P\}\ c\ \{Q\}$

Soundness and Completeness

- Soundness:

$$\vdash_p \{P\}\ c\ \{Q\} \Longrightarrow\ \models_p \{P\}\ c\ \{Q\}$$

- Relative completeness:

$$\models_p \{P\}\ c\ \{Q\} \Longrightarrow\ \vdash_p \{P\}\ c\ \{Q\}$$

  - Weakest preconditions:

    $$\vdash_p \{wp\ c\ Q\}\ c\ \{Q\}$$

Hoare Logic: Tight Specifications

- $\models_t \{P\}\ c\ \{Q\} \equiv$
  $\forall s\ h.\ (P\ s\ h \longrightarrow (Some\ (s, h), None) \notin C\ c) \wedge$
  $(\forall s'\ h'.$
  $(Some\ (s, h), Some\ (s', h')) \in C\ c \longrightarrow$
  $P\ s\ h \longrightarrow Q\ s'\ h')$

  - Error state must not be reachable

  - Partial correctness

Hoare Rules

- Allocation (records):

$$\vdash_t \{\lambda s\, h.\ (\exists\, a.\ \textit{heap-isfree}\ h\ a\ (\textit{length}\ \textit{as})) \wedge$$
$$(\forall\, a.\ (a[\mapsto]\textit{map}\ (\lambda e.\ e\ s)\ \textit{as} \mathrel{-\!\!*} P\ (s[x \mapsto a]))\ h)\}$$
$$x :== \mathrm{list}\ \textit{as}\ \{P\}$$

- Allocation (arrays):

$$\vdash_t \{\lambda s\, h.\ (\exists\, a.\ \textit{heap-isfree}\ h\ a\ (n\ s)) \wedge$$
$$(\forall\, a\ \textit{vs}.\ \textit{length}\ \textit{vs} = n\ s \longrightarrow (a[\mapsto]\textit{vs} \mathrel{-\!\!*} P\ (s[x \mapsto$$
$$a]))\ h)\}$$
$$x :== \mathrm{alloc}\ n\ \{P\}$$

Hoare Rules, cntd.

- Lookup:

$$\vdash_t \{\lambda s\, h.\ \exists v.\ (a\, s \hookrightarrow v)\ h \wedge P\ (s[x \mapsto v])\ h\}\ x :== @a\ \{P\}$$

- Mutation:

$$\vdash_t \{\lambda s.\ a\, s \mapsto - \wedge\!*\ (a\, s \mapsto v\, s\ -\!*\ P\, s)\}\ @a :== v\ \{P\}$$

- Deallocation:

$$\vdash_t \{\lambda s.\ a\, s \mapsto - \wedge\!*\ P\, s\}\ \text{dispose}\ a\ \{P\}$$

Soundness and Completeness

- $\vdash_t \{P\}\ c\ \{Q\} \Longrightarrow\ \models_t \{P\}\ c\ \{Q\}$

- $\models_t \{P\}\ c\ \{Q\} \Longrightarrow\ \vdash_t \{P\}\ c\ \{Q\}$

    - Proof: same techniques as before

The Frame Rule

- $\models \{P\}c\{Q\} \Longrightarrow \models \{P \wedge R\}c\{Q \wedge R\}$

The Frame Rule

- $\models \{P\}c\{Q\} \Longrightarrow \models \{P \wedge R\}c\{Q \wedge R\}$
- $\models \{P\}c\{Q\} \Longrightarrow \models \{P \wedge\!\!* \, R\}c\{Q \wedge\!\!* \, R\}$
  - Safety monotonicity
  - Frame property

Lacunary Heaps

- $\textit{lacunary } h \equiv \forall\, n.\ \exists\, a.\ \textit{heap-isfree } h\ a\ n$

- Every finite heap is lacunary:

  $\textit{finite } (\textit{dom } h) \Longrightarrow \textit{lacunary } h$

- Lacunarity is preserved:

  $\langle c, \textit{Some } (s, h)\rangle \longrightarrow_c \textit{Some } (s', h') \Longrightarrow$
  $\textit{lacunary } h' = \textit{lacunary } h$

Hoare Logic

- $\models_l \{P\}\ c\ \{Q\} \equiv$
  $\forall s\ h.\ lacunary\ h \longrightarrow$
  $(P\ s\ h \longrightarrow (Some\ (s, h), None) \notin C\ c) \wedge$
  $(\forall s'\ h'.\ (Some\ (s, h), Some\ (s', h')) \in C\ c \longrightarrow P\ s\ h$
  $\longrightarrow Q\ s'\ h')$

- $\vdash_l \{P\}\ c\ \{Q\}$

- $\vdash_l \{P\}\ c\ \{Q\} \Longrightarrow \models_l \{P\}\ c\ \{Q\}$

- $\models_l \{P\}\ c\ \{Q\} \Longrightarrow \vdash_l \{P\}\ c\ \{Q\}$

The Frame Rule

- $h1 \bowtie h2 \Longrightarrow$
  $(\mathit{lacunary}\ (h1 ++ h2) \longrightarrow$
  $(\mathit{Some}\ (s, h1 ++ h2), \mathit{None}) \in C\ c \longrightarrow (\mathit{Some}\ (s, h1), \mathit{None}) \in C\ c) \wedge$
  $((\mathit{Some}\ (s, h1 ++ h2), \mathit{Some}\ (s', h')) \in C\ c \longrightarrow$
  $(\mathit{Some}\ (s, h1), \mathit{None}) \in C\ c \vee$
  $(\exists h1'.\ h1' \bowtie h2 \wedge$
  $h1' ++ h2 = h' \wedge (\mathit{Some}\ (s, h1), \mathit{Some}\ (s', h1')) \in C\ c))$

- $[\![\models_l \{P\}\ c\ \{Q\};\ \mathit{ModifiedVars}\ c \natural R]\!]$
  $\Longrightarrow \models_l \{\lambda s.\ P\ s \wedge\!* R\ s\}\ c\ \{\lambda s.\ Q\ s \wedge\!* R\ s\}$

Example: In-Place List Reversal

$$
\begin{array}{l}
\text{reverse} :: \text{var} \Rightarrow \text{var} \Rightarrow \text{var} \Rightarrow \text{com} \\
\text{reverse i j k} \equiv \\
\qquad (\text{j} :== (\lambda \text{s. null})); \\
\qquad \text{while } (\lambda \text{s. s i} \neq \text{null}) \text{ do} \\
\qquad (\\
\qquad\quad (((\text{k} :== @(\lambda \text{s. Suc (s i)})); \\
\qquad\quad (@(\lambda \text{s. Suc (s i)}) :== (\lambda \text{s. s j}))); \\
\qquad\quad (\text{j} :== (\lambda \text{s. s i}))); \\
\qquad\quad (\text{i} :== (\lambda \text{s. s k})) \\
\qquad)
\end{array}
$$

In-Place List Reversal: Correctness

- Correctness theorem:

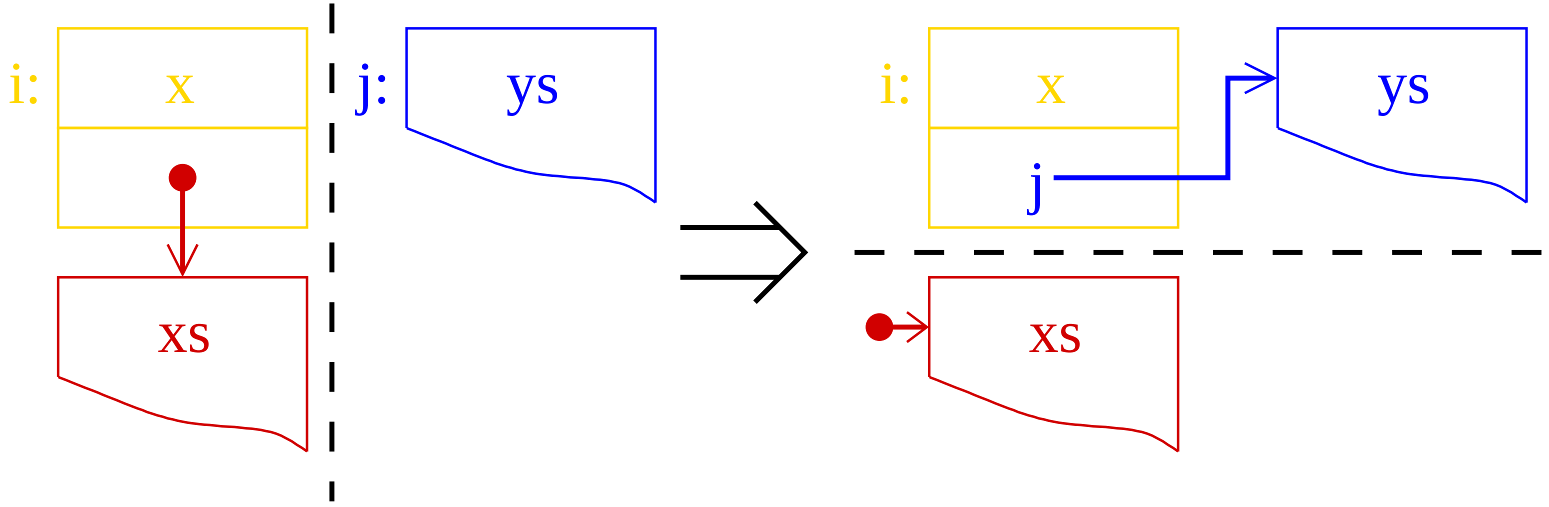
$$\models_t \{\lambda s\, h.\ \textit{heap-list vs}\ (s\ i)\ h \wedge \textit{distinct}\ [i, j, k]\}$$
$$\textit{reverse}\ i\ j\ k\ \{\lambda s.\ \textit{heap-list}\ (\textit{rev vs})\ (s\ j)\}$$

- Loop invariant:

$$\lambda s\, h.\ (\exists\, xs\ ys.$$
$$(\textit{heap-list xs}\ (s\ i) \wedge\!* \textit{heap-list ys}\ (s\ j))\ h \wedge$$
$$\textit{rev vs} = \textit{rev xs}\ @\ \textit{ys}) \wedge$$
$$\textit{distinct}\ [i, j, k]$$

In-Place List Reversal: The Proof

- $(\textit{heap-list}\ \textit{ys}\ j \wedge\!\!*\ \textit{heap-list}\ (x \# \textit{xs})\ i)\ h \Longrightarrow$
  $(\textit{heap-list}\ \textit{xs}\ (\textit{heap-lookup}\ h\ (\textit{Suc}\ i)) \wedge\!\!*\ \textit{heap-list}\ (x \# \textit{ys})\ i)$
  $(\textit{heap-update}\ h\ (\textit{Suc}\ i)\ [j])$



Conclusions

- A ready-to-use formalization of separation logic
- Meta-theoretic investigations
- Concise specifications, but less automatic proofs

Discussion

$?_*$