



Using Separation Logic to Verify Pointer Programs

Tjark Weber

webertj@in.tum.de

Technische Universität München



Overview

- The Language
- Natural Semantics
- Denotational Semantics
- Separation Logic
- Hoare Logic
- Example: In-Place List Reversal



Stores, Heaps, States

typedecI loc

types

val = nat

addr = nat

store = loc \Rightarrow val

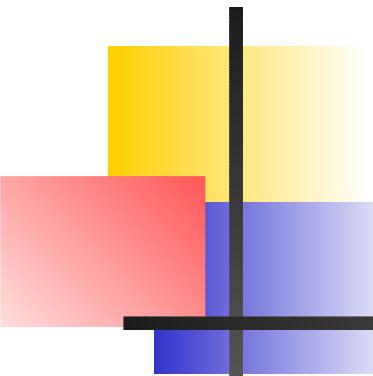
heap = addr \rightsquigarrow val

state = (store \times heap) option

aexp = store \Rightarrow val

bexp = store \Rightarrow bool





The Language: IMP ...

- SKIP
- Assign loc aexp
- Semi com com
- Cond bexp com com
- While bexp com



... with Pointers

- ListAlloc loc aexprlist
- ArrAlloc loc aexpr
- Lookup loc aexpr
- Mutate aexpr aexpr
- Dealloc aexpr



Natural Semantics: ListAlloc

■ ListAlloc:

$$\begin{aligned} & \llbracket \text{heap-isfree } h a (\text{length } as); vs = \text{map } (\lambda e. e s) as \rrbracket \\ & \implies \langle x ::= \text{list } as, \text{Some } (s, h) \rangle \xrightarrow{c} \text{Some } (s[x \mapsto a], \\ & \quad \text{heap-update } h a vs) \end{aligned}$$

■ ListAllocFail:

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


Natural Semantics: ArrAlloc

■ ArrAlloc:

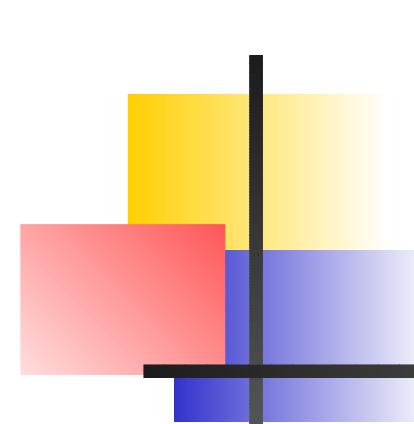
$\llbracket \text{heap-isfree } h a (n s); \text{length } vs = n s \rrbracket$

$\implies \langle x ::= \text{alloc } n, \text{Some } (s, h) \rangle \xrightarrow{c} \text{Some } (s[x \mapsto a], \text{heap-update } h a vs)$

■ ArrAllocFail:

$\forall a. \neg \text{heap-isfree } h a (n s) \implies \langle x ::= \text{alloc } n, \text{Some } (s, h) \rangle \xrightarrow{c} \text{None}$





Natural Semantics: Lookup

- Lookup:

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

- LookupFail:

$$a \ s \notin \text{dom } h \implies \langle x ::= @a, \text{Some}(s, h) \rangle \xrightarrow{c} \text{None}$$


Natural Semantics: Mutate

■ Mutate:

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

■ MutateFail:

$$a \ s \notin \text{dom } h \implies \langle @a := v, \text{Some}(s, h) \rangle \xrightarrow{c} \text{None}$$


Natural Semantics: Dealloc

■ Dealloc:

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

■ DeallocFail:

$$a \ s \notin \text{dom } h \implies \langle \text{dispose } a, \text{Some } (s, h) \rangle \xrightarrow{c} \text{None}$$


Some Semantical Properties

- We cannot recover from an abort:

$$\langle c, \text{None} \rangle \longrightarrow_c s = (s = \text{None})$$

- Execution is not deterministic:

$$\exists c s t t'. \langle c, s \rangle \longrightarrow_c \text{Some } t \wedge \langle c, s \rangle \longrightarrow_c \text{Some } t' \wedge t \neq t'$$

$$\exists c s s'. \langle c, s \rangle \longrightarrow_c \text{Some } s' \wedge \langle c, s \rangle \longrightarrow_c \text{None}$$



Denotational Semantics

■ C_ListAlloc:

$$\begin{aligned} C(x ::= \text{list } as) = \\ \{ & (\text{Some}(s, h), \text{Some}(s[x \mapsto a], \text{heap-update } h a (\text{map } (\lambda e. e s) as))) \mid s h a. \text{heap-isfree } h a (\text{length } as) \} \cup \\ \{ & (\text{Some}(s, h), \text{None}) \mid s h. \forall a. \neg \text{heap-isfree } h a (\text{length } as) \} \\ \cup & \{ (\text{None}, \text{None}) \} \end{aligned}$$

■ Equivalence of denotational and natural semantics:

$$((s, t) \in C c) = \langle c, s \rangle \xrightarrow{c} t$$



Separation Logic

- $\wedge, \vee, \neg, \rightarrow, \dots$

- 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:

$$(P \rightarrow^* Q) h \equiv \forall h'. h' \bowtie h \wedge P h' \rightarrow Q (h ++ h')$$



Basic Assertions

- $\text{emp } h \equiv \text{dom } h = \{\}$
- $(a \mapsto v) \ h \equiv \text{dom } h = \{a\} \wedge \text{heap-lookup } h \ a = v$
- $(a \mapsto -) \ h \equiv \exists v. \ (a \mapsto v) \ h$
- $a \leftarrow v \equiv a \mapsto v \wedge^* \text{true}$

Properties of $\wedge*$

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



Properties of $\wedge*$, cntd.

- $((\lambda h. P h \vee Q h) \wedge* R) h = (P \wedge* R) h \vee (Q \wedge* R) h$
- $((\lambda h. P h \wedge Q h) \wedge* R) h \longrightarrow (P \wedge* R) h \wedge (Q \wedge* R) h$
- $((\lambda h. \exists x. P x h) \wedge* Q) h = (\exists x. (P x \wedge* Q) h)$
- $((\lambda h. \forall x. P x h) \wedge* Q) h \longrightarrow (\forall x. (P x \wedge* Q) h)$



Properties of \wedge^* and

\longrightarrow^*

- $\llbracket \forall h. P h \longrightarrow P' h; \forall h. Q h \longrightarrow Q' h \rrbracket \implies (P \wedge^* Q)$
 $h \longrightarrow (P' \wedge^* Q') h$
- $\forall h. (P \wedge^* Q) h \longrightarrow R h \implies Ph \longrightarrow (Q \longrightarrow^* R) h$
- $\forall h. Ph \longrightarrow (Q \longrightarrow^* R) h \implies (P \wedge^* Q) h \longrightarrow Rh$

Pure Assertions

- $\text{pure } P \equiv \forall h h'. P h = P h'$
- $\llbracket \text{pure } P; \text{pure } Q \rrbracket \implies (P h \wedge Q h) = (P \wedge^* Q) h$
- $\llbracket \text{pure } P; \text{pure } Q \rrbracket \implies (P \longrightarrow^* Q) h = (P h \longrightarrow Q h)$

Intuitionistic Assertions

- *intuitionistic P* $\equiv \forall h h'. P h \longrightarrow h \subseteq_m h' \longrightarrow P h'$
- *intuitionistic P* $\implies (P \wedge^* \text{true}) h \longrightarrow P h$
- *intuitionistic P* $\implies P h \longrightarrow (\text{true} \longrightarrow^* P) h$



Domain-exact Assertions

- *domain-exact P* $\equiv \forall h h'. Ph \longrightarrow Ph' \longrightarrow \text{dom } h = \text{dom } h'$
- *domain-exact R* $\implies (P \wedge^* R) h \wedge (Q \wedge^* R) h \longrightarrow ((\lambda h. Ph \wedge Qh) \wedge^* R) h$
- *domain-exact Q* $\implies (\forall x. (Px \wedge^* Q) h) \longrightarrow ((\lambda h. \forall x. Pxh) \wedge^* Q) h$



Hoare Logic

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

- No error state reachable

- Partial correctness



Hoare Rules

■ ListAlloc:

$$\vdash \{\lambda s h. (\exists a. \text{heap-isfree } h a (\text{length } as)) \wedge (\forall a. (a[\mapsto] \text{map } (\lambda e. e s) as \longrightarrow^* P (s[x \mapsto a])) h)\} \\ x ::= \text{list as } \{P\}$$

■ ArrAlloc:

$$\vdash \{\lambda s h. (\exists a. \text{heap-isfree } h a (n s)) \wedge (\forall a vs. \text{length } vs = n s \longrightarrow (a[\mapsto] vs \longrightarrow^* P (s[x \mapsto a])) h)\} \\ x ::= \text{alloc } n \{P\}$$


Hoare Rules, cntd.

■ Lookup:

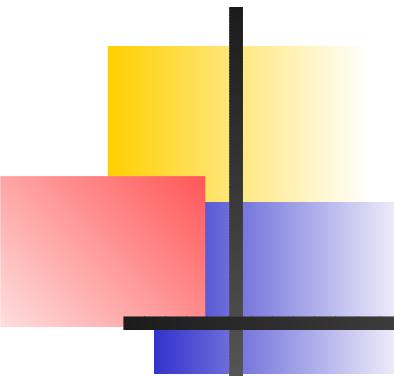
$$\vdash \{\lambda s h. \exists v. (a \ s \hookrightarrow v) \ h \wedge P(s[x \mapsto v]) \ h\} \ x := @_a \{P\}$$

■ Mutate:

$$\vdash \{\lambda s. a \ s \mapsto - \wedge^* (a \ s \mapsto v \ s \longrightarrow^* P \ s)\} @_a := v \ {P\}$$

■ Dealloc:

$$\vdash \{\lambda s. a \ s \mapsto - \wedge^* P \ s\} \ dispose \ a \ {P\}$$

Soundness and Completeness

- Soundness:

$$\vdash \{P\} c \{Q\} \implies \models \{P\} c \{Q\}$$

- Relative completeness:

$$\models \{P\} c \{Q\} \implies \vdash \{P\} c \{Q\}$$

- Weakest precondition:

$$\vdash \{wp c Q\} c \{Q\}$$



Example: In-place List Reversal

```
reverse :: loc ⇒ loc ⇒ loc ⇒ com
reverse i j k == (j == (λs. 0));
                  while (λs. s i ≠ 0) do
                    (
                      (((k == @(\s. Suc (s i)));
                        (@(\s. Suc (s i)) == (λs. s j)));
                        (j == (λs. s i)));
                        (i == (λs. s k)))
                    )
```



In-place List Reversal, cndt.

theorem reverse-correct:

$$\begin{aligned} & \vdash \{ \lambda s h. \text{heap-list } vs (s i) h \wedge i \neq j \wedge i \neq k \wedge j \neq k \} \\ & \quad \text{reverse } i j k \\ & \{ \lambda s h. \text{heap-list } (\text{rev } vs) (s j) h \} \end{aligned}$$

■ Loop invariant:

$$\begin{aligned} & \lambda s h. (\exists xs ys. \\ & \quad (\text{heap-list } xs (s i) \wedge * \text{heap-list } ys (s j)) h \wedge \\ & \quad \text{rev } vs = \text{rev } xs @ ys) \wedge \\ & \quad i \neq j \wedge i \neq k \wedge j \neq k \end{aligned}$$



List Reversal without Separation Logic

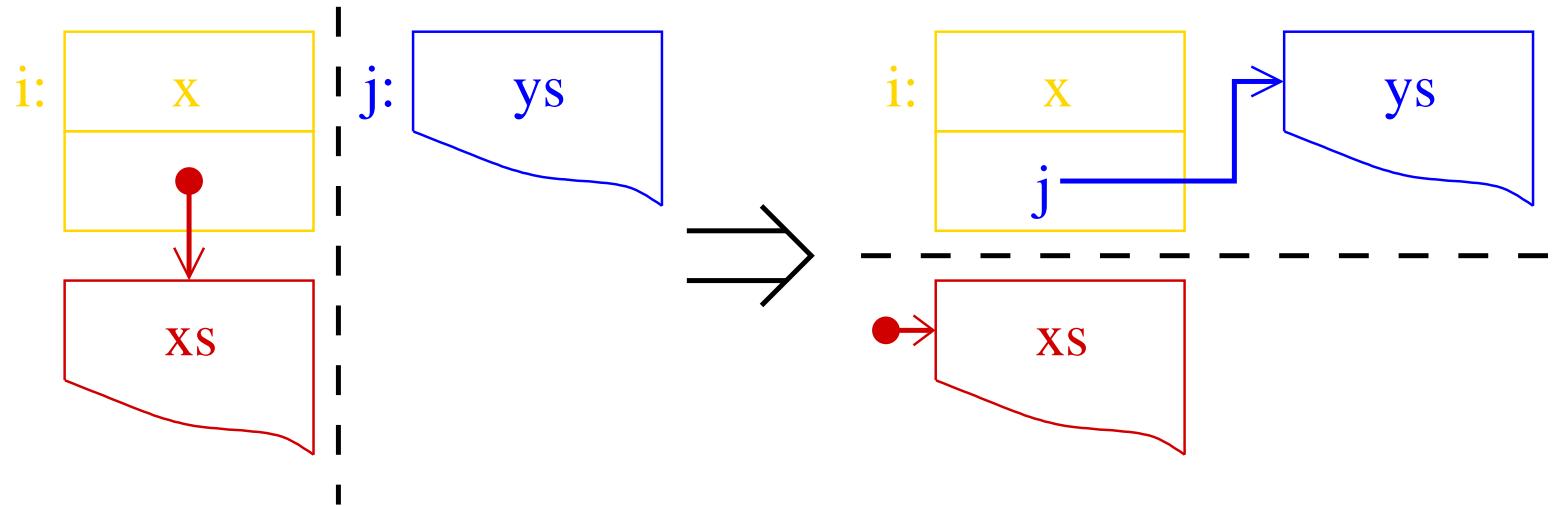
- $\lambda s h. (\exists xs ys.$
 $(\text{heap-list } xs (s i) \wedge* \text{heap-list } ys (s j)) h \wedge$
 $\text{rev } vs = \text{rev } xs @ ys) \wedge$
 $i \neq j \wedge i \neq k \wedge j \neq k$

- $\exists As' Bs'.$
 $\text{list } tl p As' \wedge$
 $\text{list } tl q Bs' \wedge \text{set } As' \cap \text{set } Bs' = \{\} \wedge \text{rev } As' @ Bs' = \text{rev } As$
 $@ Bs$



List Reversal: The Proof

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



Conclusions and Future Work

- No complete axiom scheme for \wedge^* and \longrightarrow^*
- Syntactic sugar
- Frame rule
- Annotations, Verification Condition Generator
- Hoare rules for total correctness



Discussion

? *

