

Hoare logic and Model checking

Revision class

Rini Banerjee & Hiroyuki Katsura rb2018, hk590

University of Cambridge

CST Part II – 2025/26

Hoare logic and separation logic

The concept of ownership

Ownership of a heap cell is the permission to safely read/write/dispose of it. **This ownership is not duplicable.**

E.g.: use-after-free: $\text{dispose}(X); [X] := 5$

Separation logic:

$\{X \mapsto v\}$
 $\text{dispose}(X);$
 $\{emp\}$
proof fails
 $\{X \mapsto v\}$
 $[X] := 5$
 $\{X \mapsto 5\}$

If ownership were duplicable:

$\{X \mapsto v\}$
 $\{X \mapsto v * X \mapsto v\}$
 $\text{dispose}(X);$
 $\{X \mapsto v\}$
 $[X] := 5$
 $\{X \mapsto 5\}$

Pure assertions

$$\llbracket - \rrbracket (=) : \text{Assertion} \rightarrow \text{Stack} \rightarrow \mathcal{P}(\text{Heap})$$

$$\llbracket \perp \rrbracket (s) \stackrel{\text{def}}{=} \emptyset$$

$$\llbracket \top \rrbracket (s) \stackrel{\text{def}}{=} \text{Heap}$$

$$\llbracket P \wedge Q \rrbracket (s) \stackrel{\text{def}}{=} \llbracket P \rrbracket (s) \cap \llbracket Q \rrbracket (s)$$

$$\llbracket P \vee Q \rrbracket (s) \stackrel{\text{def}}{=} \llbracket P \rrbracket (s) \cup \llbracket Q \rrbracket (s)$$

$$\llbracket P \Rightarrow Q \rrbracket (s) \stackrel{\text{def}}{=} \{h \in \text{Heap} \mid h \in \llbracket P \rrbracket (s) \Rightarrow h \in \llbracket Q \rrbracket (s)\}$$

⋮

What is the meaning of pure assertions, such as \top or $t_1 = t_2$? Do they implicitly require the heap to be empty?

Semantics of pure assertions

$\llbracket - \rrbracket (=) : \textit{Assertion} \rightarrow \textit{Stack} \rightarrow \mathcal{P}(\textit{Heap})$

$$\llbracket t_1 = t_2 \rrbracket (s) = \{h \mid \llbracket t_1 \rrbracket (s) = \llbracket t_2 \rrbracket (s)\} = \begin{cases} \textit{Heap} & \text{if } \llbracket t_1 \rrbracket (s) = \llbracket t_2 \rrbracket (s) \\ \emptyset & \text{otherwise} \end{cases}$$

More generally, the semantics of a pure assertion in a stack s :

Informally: “check the pure assertion in s ”; if it holds in s , return the set of all heaps, if not return the empty set of heaps.

Formally: don't worry about it, because we have not defined it.

Semantics of pure assertions, wrt. heap (continued).

The 2019 exam paper 8, question 7 asks:

$$\{N = n \wedge N \geq 0\}$$
$$X := \text{null}; \text{ while } N > 0 \text{ do } (X := \text{alloc}(N, X); N := N - 1)$$
$$\{\text{list}(X, [1, \dots, n])\}$$

(I have not checked whether that year used different definitions from ours, but) **This seems to be missing emp in the pre-condition:** $\{N = n \wedge N \geq 0 \wedge \text{emp}\}$

Why? $\{N = n \wedge N \geq 0\}$ makes no statement about the heap — if the stack has the right property, it is satisfied by any heap. But without the emp requirement, we would not be able to prove the post-condition $\{\text{list}(X, [1, \dots, n])\}$, which asserts that the **only** ownership is that of the list predicate instance.

Another error

Related: error in 2021 Paper 8 Question 8.

The pre-condition should have

$$\dots \wedge 1 \leq S$$

instead of

$$\dots * 1 \leq S$$

.

Conjunction and separating conjunction

What are the differences between them and when to use which?
And how do they interact with pure assertions?

$$\begin{aligned} \llbracket P * Q \rrbracket(s) &\stackrel{\text{def}}{=} \left\{ h \in \text{Heap} \left| \begin{array}{l} \exists h_1, h_2. \quad h_1 \in \llbracket P \rrbracket(s) \wedge \\ \quad \quad \quad h_2 \in \llbracket Q \rrbracket(s) \wedge \\ \quad \quad \quad h = h_1 \uplus h_2 \end{array} \right. \right\} \\ \llbracket P \wedge Q \rrbracket(s) &\stackrel{\text{def}}{=} \llbracket P \rrbracket(s) \cap \llbracket Q \rrbracket(s) \end{aligned}$$

Conjunction and separating conjunction (continued)

$$\llbracket P * Q \rrbracket(s) \stackrel{\text{def}}{=} \left\{ h \in \text{Heap} \left| \begin{array}{l} \exists h_1, h_2. \quad h_1 \in \llbracket P \rrbracket(s) \wedge \\ \quad h_2 \in \llbracket Q \rrbracket(s) \wedge \\ \quad h = h_1 \uplus h_2 \end{array} \right. \right\}$$

$$\llbracket P \wedge Q \rrbracket(s) \stackrel{\text{def}}{=} \llbracket P \rrbracket(s) \cap \llbracket Q \rrbracket(s)$$

$p_1 \mapsto v_1 * p_2 \mapsto v_2$ **vs.** $p_1 \mapsto v_1 \wedge p_2 \mapsto v_2$

- $p_1 \mapsto v_1 * p_2 \mapsto v_2$ holds for a heap h that is the disjoint union of heaplets h_1 and h_2 , where h_1 contains just cell p_1 , with value v_1 , and h_2 just cell p_2 , with value v_2 . So: ownership of **two disjoint** heap cells p_1 and p_2 with $p_1 \neq p_2$.
- $p_1 \mapsto v_1 \wedge p_2 \mapsto v_2$ holds for a heap h that satisfies two assertions simultaneously (is in the intersection of their interpretations):
 - (1) $p_1 \mapsto v_1$: h is a heap of just one heap cell, p_1 with value v_1
 - (2) $p_2 \mapsto v_2$: h is a heap of just one heap cell, p_2 with value v_2So: ownership of just **one** heap cell, $p_1 = p_2$ with value $v_1 = v_2$.

Conjunction and separating conjunction (continued)

$$\llbracket P * Q \rrbracket(s) \stackrel{\text{def}}{=} \left\{ h \in \text{Heap} \mid \begin{array}{l} \exists h_1, h_2. \\ h_1 \in \llbracket P \rrbracket(s) \wedge \\ h_2 \in \llbracket Q \rrbracket(s) \wedge \\ h = h_1 \uplus h_2 \end{array} \right\}$$

$$\llbracket P \wedge Q \rrbracket(s) \stackrel{\text{def}}{=} \llbracket P \rrbracket(s) \cap \llbracket Q \rrbracket(s)$$

$(p \mapsto 1) * Y = 0$ vs. $(p \mapsto 1) \wedge Y = 0$

- $(p \mapsto 1) * Y = 0$ holds for a stack s and a heap h where h is the disjoint union of heaplets h_1 and h_2 , such that h_1 contains ownership of one cell, p with value 1, and h_2 is an arbitrary heap if s satisfies $Y = 0$. So, s must map Y to 0 and h is the disjoint union of the heaplet of just p with value 1 and **an arbitrary disjoint** heap h_2 .
- $(p \mapsto 1) \wedge Y = 0$ holds for a stack s and a heap h satisfying two assertion simultaneously: $p \mapsto 1$ and $Y = 0$. This means s must map Y to 0 and h must be the heap consisting of just that one cell.

emp in the alloc rule

Q: Why have the 'emp' in the precondition of the alloc rule?

$$\frac{}{\vdash \{X = x \wedge emp\} X := \mathbf{alloc}(E_0, \dots, E_n) \{X \mapsto E_0[x/X], \dots, E_n[x/X]\}}$$

A: This is needed for soundness. Otherwise the alloc rule would allow us to silently drop ownership of other heap cells.

Use of frame rule (to obtain “ $\dots \wedge emp$ ”) in Lecture 5, slide 28

$$\{list(Y, \alpha) \wedge X = x\}$$

$$\{\exists z. (list(Y, \alpha) \wedge X = x) \wedge HEAD = z\}$$

$$\{(list(Y, \alpha) \wedge X = x) \wedge HEAD = z\}$$

$$\{(list(Y, \alpha) \wedge X = x) * (HEAD = z \wedge emp)\}$$

$$\{HEAD = z \wedge emp\}$$

HEAD := alloc(X, Y)

$$\{HEAD \mapsto X[z/HEAD], Y[z/HEAD]\}$$

$$\{HEAD \mapsto X, Y\}$$

$$\{(list(Y, \alpha) \wedge X = x) * HEAD \mapsto X, Y\}$$

$$\{(list(Y, \alpha) * HEAD \mapsto X, Y) \wedge X = x\}$$

$$\{\exists z. (list(Y, \alpha) * HEAD \mapsto X, Y) \wedge X = x\}$$

$$\{(list(Y, \alpha) * HEAD \mapsto X, Y) \wedge X = x\}$$

It is good to be careful of the possibly unexpected behaviour of the new separation logic assertions!

Example: 2019-p08-q07, e

Give a loop invariant for the following list concatenation triple:

```
{list(X,  $\alpha$ ) * list(Y,  $\beta$ )}
if X = null then
  Z:=Y
else (
  Z := X; U := Z; V := [Z + 1];
  while V  $\neq$  null do (U := V ; V := [V + 1]);
  [U + 1] := Y
)
{list(Z,  $\alpha ++ \beta$ )}
```

Example: 2019-p08-q07, e

$\{\text{list}(X, \alpha) * \text{list}(Y, \beta)\}$

if $X = \text{null}$ then

$Z := Y$

else (

$Z := X; U := Z; V := [Z + 1];$

while $V \neq \text{null}$ do ($U := V; V := [V + 1];$

$[U + 1] := Y$

)

$\{\text{list}(Z, \alpha ++ \beta)\}$

Example: 2019-p08-q07, e

$\{(\text{list}(X, \alpha) * \text{list}(Y, \beta)) \wedge X \neq \text{null}\}$

$Z := X; U := Z; V := [Z + 1];$

$\text{while } V \neq \text{null} \text{ do } (U := V ; V := [V + 1]);$

$[U + 1] := Y$

$\{\text{list}(Z, \alpha ++ \beta)\}$

$\{\text{list}(X, \alpha) * \text{list}(Y, \beta)\} \wedge X \neq \text{null}\}$

$\{\exists t, p, \delta. \alpha = [t] \uparrow \delta \wedge (X \mapsto t, p * \text{list}(p, \delta) * \text{list}(Y, \beta))\}$

$Z := X;$

$\{\exists t, p, \delta. \alpha = [t] \uparrow \delta \wedge (Z \mapsto t, p * \text{list}(p, \delta) * \text{list}(Y, \beta))\}$

$U := Z;$

$\{\exists t, p, \delta. \alpha = [t] \uparrow \delta \wedge U = Z \wedge (Z \mapsto t, p * \text{list}(p, \delta) * \text{list}(Y, \beta))\}$

$V := [Z + 1];$

$\{\exists t, \delta. \alpha = [t] \uparrow \delta \wedge U = Z \wedge (Z \mapsto t, V * \text{list}(V, \delta) * \text{list}(Y, \beta))\}$

$I : \{\exists \gamma, t, \delta. \alpha = \gamma \uparrow [t] \uparrow \delta \wedge (\text{plist}(Z, \gamma, U) * \text{plist}(U, [t], V) * \text{list}(V, \delta) * \text{list}(Y, \beta))\}$

while $V \neq \text{null}$ do ($U := V ; V := [V + 1]$);

$\{\exists \gamma, t, \delta. \alpha = \gamma \uparrow [t] \uparrow \delta \wedge (\text{plist}(Z, \gamma, U) * \text{plist}(U, [t], V) * \text{list}(V, \delta) * \text{list}(Y, \beta))$
 $\wedge \neg(V \neq \text{null})\}$

$[U + 1] := Y$

$\{\exists \gamma, t, \delta. \alpha = \gamma \uparrow [t] \uparrow \delta \wedge (\text{plist}(Z, \gamma, U) * \text{plist}(U, [t], Y) * \text{list}(V, \delta) * \text{list}(Y, \beta))$
 $\wedge \neg(V \neq \text{null})\}$

$\{\text{list}(Z, \alpha \uparrow \beta)\}$

Proof outlines + loop invariants

Q: How much detail to give in proof outline in exam?

Q: If asked to provide a loop invariant, do you need to provide the full proof?

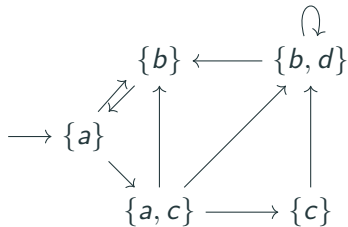
A: The exam text will be clear about that.

Model Checking

Temporal operators, e.g. in CTL

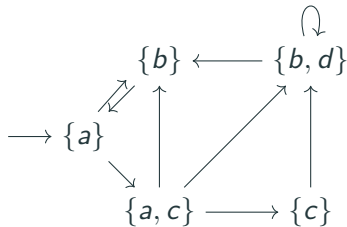
- $AX\psi$ and $EX\psi$:
 - Does the state satisfying ψ have to be different from the starting state?
 - Does ψ have to continue holding?
- $A(\psi_1 U \psi_2)$ and $E(\psi_1 U \psi_2)$:
 - Does ψ_1 have to continue holding?
 - What about ψ_2 ?

LTL examples



ϕ	$M \models \phi$
a	yes
Xa	no
Fb	yes
Fc	no
$(a \vee b)Uc$	no
dUa	yes
$G(a \vee b \vee c)$	yes
GFb	yes
FGb	no

CTL examples



ψ	$M \models \psi$
$EX(b \wedge \neg c)$	yes
AFd	no
EFd	yes
$E(aUd)$	yes
$AGEFd$	yes
$AFEGd$	no
$EFEGd$	yes
$E((a \vee c)U(EGb))$	yes

LTL/CTL expressivity

An elevator property: **“If it is possible to answer a call to some level in the next step, then the elevator does that”**

CTL formula ψ : $A\ G\ ((Call_2 \wedge E\ X\ Loc_2) \rightarrow A\ X\ Loc_2)$

Q: Can we express the same in LTL with formula ϕ :
 $G\ (Call_2 \wedge (Loc_1 \vee Loc_3)) \rightarrow X\ Loc_2?$

This depends on the details of the elevator temporal model.¹ In any case, ψ and ϕ are not generally equivalent. The point is: expressing properties of the tree of possible paths out of a given state — such as asserting the **existence** of some path — is not possible with LTL.

¹I think — the way we have sketched the elevator in lecture 7 — this will not work: $Loc_1 \vee Loc_3$ does not imply there exists a next step such that Loc_2 holds.

LTL/CTL expressivity

An LTL formula not expressible in CTL: $\phi = (F p) \rightarrow (F q)$.

a) CTL formula $\psi_1 = (A F p) \rightarrow (A F q)$.

ϕ does not hold, ψ_1 does.



b) CTL formula $\psi_2 = A G (p \rightarrow (A F q))$.

ϕ holds, ψ_2 does not.

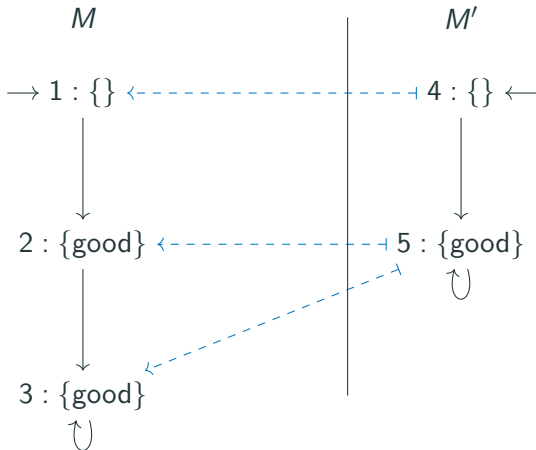


It is good to be careful about the unexpected interaction of the temporal operators, with other temporal operators and with path quantifiers.

Simulation relations

Q: Why simulation relations and not simulation functions?

Example: $AP = AP' = \{\text{good}\}$. M simulates M'



Good luck!