Hoare logic and Model checking

Revision class

Christopher Pulte cp526
University of Cambridge

CST Part II – 2021/22

Hoare logic and separation logic

The concept of ownership

Ownership of a heap cell is the permission to (safely) read/write/dispose of it: $\{P\}$ C $\{Q\}$ guarantees C does not fail, the ownership of the locations in P is sufficient.

Essential: this ownership is not duplicable.

The concept of ownership (continued)

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

Separation logic:	**If ownership was duplicable:**
$\{X \mapsto v\}$	$\{X \mapsto v\}$
dispose(X);	$\{X \mapsto v * X \mapsto v\}$
$\{emp\}$	dispose(X);
proof fails	$\{X \mapsto v\}$
$\{X \mapsto v\}$	[X] := 5
[X] := 5	$\{X \mapsto 5\}$
$\{X \mapsto 5\}$	

(This is very different from Hoare logic assertions that are freely duplicable.)

Ownership formally, and linear vs. affine

$$[\![-]\!](=) : \mathit{Assertion} \to \mathit{Stack} \to \mathcal{P}(\mathit{Heap})$$

$$[\![t_1 \mapsto t_2]\!](s) \stackrel{\mathit{def}}{=} \left\{ h \in \mathit{Heap} \;\middle|\; \exists \ell, N. \begin{array}{l} [\![t_1]\!](s) = \ell \wedge \\ \ell \neq \mathbf{null} \wedge \\ [\![t_2]\!](s) = N \wedge \\ \mathit{dom}(h) = \{\ell\} \wedge \\ h(\ell) = N \end{array} \right\}$$

$t_1 \mapsto t_2$ asserts ownership of location ℓ, so to capture ownership, requires $\{\ell\} \subseteq \mathit{dom}(h)$.

- In our **linear** separation logic resources **cannot be dropped**: to prevent memory leaks, we require $\mathit{dom}(h) = \{\ell\}$.
- Having the requirement $\{\ell\} \subseteq \mathit{dom}(h)$ instead would give us an affine separation logic.

Memory leaks?

Ok in an **affine** logic.

$\{X \mapsto 1 * Y \mapsto 2\}$
skip
$\{X \mapsto 1 * Y \mapsto 2\}$
$\{X \mapsto 1\}$

We use a **linear** logic.

$\{X \mapsto 1 * Y \mapsto 2\}$
dispose(Y);
$\{X \mapsto 1\}$

How is ownership related to framing?

If we have proved $\{P\}\ C\ \{Q\}$ for some program C and we want to use this triple in a proof involving assertion R, we can use the frame rule to conclude $\{P * R\}\ C\ \{Q * R\}$: R is preserved by C.

$$\frac{\vdash \{P\}\ C\ \{Q\} \qquad \mathit{mod}(C) \cap \mathit{FV}(R) = \emptyset}{\vdash \{P * R\}\ C\ \{Q * R\}}$$

Intuitively: P must have all the ownership required for the safe execution of C — all the parts of the heap that C manipulates. The separating conjunction ensures that R cannot have ownership of those heap locations (or the precondition is false).

Recall: $P * R$ requires the disjointness of the heap cells for which P and R assert ownership.

Formal semantics of separation logic triples

Written formally, the semantics is:

$$\models \{P\}\ C\ \{Q\} \stackrel{\mathit{def}}{=}$$
$$\forall s, h_1, h_F.\ \mathit{dom}(h_1) \cap \mathit{dom}(h_F) = \emptyset \wedge h_1 \in [\![P]\!](s) \Rightarrow$$
$$\left(\begin{array}{l} (\neg(\langle C, \langle s, h_1 \uplus h_F \rangle\rangle \to^* \lightning)) \wedge \\ \left(\forall s', h'.\ \begin{array}{l} \langle C, \langle s, h_1 \uplus h_F \rangle\rangle \to^* \langle \mathbf{skip}, \langle s', h' \rangle\rangle \Rightarrow \\ \exists h_1'.\ h' = h_1' \uplus h_F \wedge h_1' \in [\![Q]\!](s') \end{array} \right) \end{array} \right)$$

This has "framing baked in". Q: Does it have to?

No. See for instance: "Separation Logic: A Logic for Shared Mutable Data Structures", J. C. Reynolds; and "A Semantic Basis for Local Reasoning.", H. Yang and P. O'Hearn

Pure assertions

$$
\begin{aligned}
[\![-]\!](=) &: \mathit{Assertion} \to \mathit{Stack} \to \mathcal{P}(\mathit{Heap}) \\
[\![\bot]\!](s) &\stackrel{\mathit{def}}{=} \emptyset \\
[\![\top]\!](s) &\stackrel{\mathit{def}}{=} \mathit{Heap} \\
[\![P \wedge Q]\!](s) &\stackrel{\mathit{def}}{=} [\![P]\!](s) \cap [\![Q]\!](s) \\
[\![P \vee Q]\!](s) &\stackrel{\mathit{def}}{=} [\![P]\!](s) \cup [\![Q]\!](s) \\
[\![P \Rightarrow Q]\!](s) &\stackrel{\mathit{def}}{=} \{h \in \mathit{Heap} \mid h \in [\![P]\!](s) \Rightarrow h \in [\![Q]\!](s)\} \\
&\vdots
\end{aligned}
$$

What is the meaning of pure assertion $X = Y$?

$$[\![X = Y]\!](s) = \{h \mid s(X) = s(Y)\} = \begin{cases} \mathit{Heap} & \text{if } [\![X]\!](s) = [\![Y]\!](s) \\ \emptyset & \text{otherwise} \end{cases}$$

Semantics of pure assertions

$$[\![X = Y]\!](s) = \{h \mid s(X) = s(Y)\} = \begin{cases} \mathit{Heap} & \text{if } [\![X]\!](s) = [\![Y]\!](s) \\ \emptyset & \text{otherwise} \end{cases}$$

$$[\![p(t_1, \ldots t_n]\!](s) = \{h \mid [\![p]\!]([\![t_1]\!](s), \ldots, [\![t_n]\!](s))\}$$

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

Do pure assertions such as $X = 1$ or $X = Y$ assert properties about the heap? E.g. do they implicitly assert $\cdots \wedge \text{emp}$ (ownership of the empty resource/heap)? No.

The meaning of $\top$, for instance, is $[\![\top]\!](s) = \mathit{Heap}$, the set of all heaps (not the set containing the empty heap).

Semantics of pure assertions, wrt. heap (continued)

The 2019 exam paper 8, question 7 asks:

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

(I have not checked whether that year used different definitions from ours, but) **This does seem to be missing the 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 — the precondition is satisfied by any heap (and suitable stack).

But without the emp requirement, we would not be able prove the post-condition $\text{list}(1, \ldots, n)$, which asserts that the **only** ownership is that of the list predicate instance.

Conjunction and separating conjunction

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

$$[\![P * Q]\!](s) \stackrel{def}{=} \left\{ h \in Heap \,\middle|\, \exists h_1, h_2. \begin{array}{l} h_1 \in [\![P]\!](s) \wedge \\ h_2 \in [\![Q]\!](s) \wedge \\ h = h_1 \uplus h_2 \end{array} \right\}$$

$$[\![P \wedge Q]\!](s) \stackrel{def}{=} [\![P]\!](s) \cap [\![Q]\!](s)$$

Conjunction and separating conjunction (continued)

$$[\![P * Q]\!](s) \stackrel{def}{=} \left\{ h \in Heap \,\middle|\, \exists h_1, h_2. \begin{array}{l} h_1 \in [\![P]\!](s) \wedge \\ h_2 \in [\![Q]\!](s) \wedge \\ h = h_1 \uplus h_2 \end{array} \right\}$$

$$[\![P \wedge Q]\!](s) \stackrel{def}{=} [\![P]\!](s) \cap [\![Q]\!](s)$$

$p_1 \mapsto v_1 * p_2 \mapsto v_2$ **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_2
  So: ownership of just **one** heap cell, $p_1 = p_2$ with value $v_1 = v_2$.

Conjunction and separating conjunction (continued)

$$[\![P * Q]\!](s) \stackrel{def}{=} \left\{ h \in Heap \,\middle|\, \exists h_1, h_2. \begin{array}{l} h_1 \in [\![P]\!](s) \wedge \\ h_2 \in [\![Q]\!](s) \wedge \\ h = h_1 \uplus h_2 \end{array} \right\}$$

$$[\![P \wedge Q]\!](s) \stackrel{def}{=} [\![P]\!](s) \cap [\![Q]\!](s)$$

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

It is good to be careful about the unexpected interaction of the usual logical connectives with the new separation logic connectives!

Variable assignment, heap derefencing, heap assignment

Variable assignment $\overline{\vdash \{P[E/X]\}\ X := E\ \{P\}}$

Heap assignment $\overline{\vdash \{E_1 \mapsto t\}\ [E_1] := E_2\ \{E_1 \mapsto E_2\}}$

Heap derefence $\overline{\vdash \{E \mapsto v \wedge X = x\}\ X := [E]\ \{E[x/X] \mapsto v \wedge X = v\}}$

Why do the rules look so different? Could they be made more similar?

1. $X := E$ and $[X] := E$ are fundamentally different operations.
2. A heap assignment rule with substitution behaviour (similar to variable assignment) would not work: there is nothing to be substituted, since E_1 is a **pointer**.
3. One could have a separation logic with **ownership of program variables**, where variable assignment might look more similar to heap assignment.
4. One could indeed have a variable assignment rule more similar to (the "variable-updating" part of) heap dereferencing

Proof outlines to proof trees

Good strategy for converting proof outlines to proof trees: read "inside out", starting with the inner triples around commands. Note: these steps work only if it is a **detailed** proof outline – with all the steps.

- $\{P\}\ C\ \{Q\}$, an inner triple for an "atomic command" (skip, assignment, heap derefence, heap assignment, allocation, disposal), translates to an application of the Hoare/separation logic inference rule for that command C.

Proof outlines to proof trees (continued)

- $\{P_1\}$
    $\{P_2\}$
    C
    $\{Q_2\}$
  $\{Q_1\}$

  The rule for existentials and the frame rule are indicated by indentation. (Which of these should be clear from the outline.) This translates to an instance of either of these:

$$\frac{\{P_2\}C\{Q_2\} \qquad \text{side condition} \ldots}{\{P_1\}C\{Q_1\}}$$

Proof outlines to proof trees (continued)

- $\{P_1\}$
  $\{P_2\}$
  C
  $\{Q_2\}$
  $\{Q_1\}$

  The rule of consequence is indicated by un-indented brackets of assertions

$$\frac{\vdash_{FOL} P_1 \Rightarrow P_2 \qquad \vdash \{P_2\}\ C\ \{Q_2\} \qquad \vdash_{FOL} Q_2 \Rightarrow Q_1}{\vdash \{P_1\}\ C\ \{Q_1\}}$$

For an example of how to read proof outlines, see lecture 5, slide 10 (and video). Note that the website has updated slides for these compared to the printed handout.

Proof outlines

How much detail to give in proof outline in exam?

Model Checking

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: $\psi = \mathsf{A}\ \mathsf{G}\ ((\mathsf{Call}_2 \wedge \mathsf{E}\ \mathsf{X}\ \mathsf{Loc}_2) \rightarrow \mathsf{A}\ \mathsf{X}\ \mathsf{Loc}_2)$

Q: Can we express the same in LTL with
$\phi = \mathsf{G}\ (\mathsf{Call}_2 \wedge (\mathsf{Loc}_1 \vee \mathsf{Loc}_3)) \rightarrow \mathsf{X}\ \mathsf{Loc}_2$?

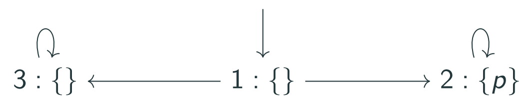
This depends on the details of the elevator temporal model this may produce the same answers.[1] In any case, ψ and ϕ are not generally equivalent. The point is: expressing properties of the tree of possible transitions out of a given state — such as asserting the **existence** of some path — is not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

[1] I think — the way we have sketched the elevator in lecture 7 — it will not: $\mathsf{Loc}_1 \vee \mathsf{Loc}_3$ does not imply there exists a next step such that Loc_2 holds.

LTL/CTL expressivity

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

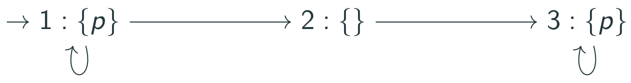
a) CTL formula $\psi_1 = (\mathsf{A}\ \mathsf{F}\ p) \rightarrow (\mathsf{A}\ \mathsf{F}\ q)$.
ϕ does not hold, ψ_1 does.



b) CTL formula $\psi_2 = \mathsf{A}\ \mathsf{G}\ (p \rightarrow (\mathsf{A}\ \mathsf{F}\ q))$.
ϕ holds, ψ_2 does not.

LTL/CTL expressivity

Why are F G p in LTL and A F A G p in CTL not equivalent?



Two kinds of infinite paths: (L1) loop in 1 forever, (L2) loop in 3 forever. Both kinds of paths **eventually** reach a state in which p holds **generally** (1 or 3, respectively). So F G p holds.

Informally: A F A G p holds if (check CTL (CTL*) semantics for):

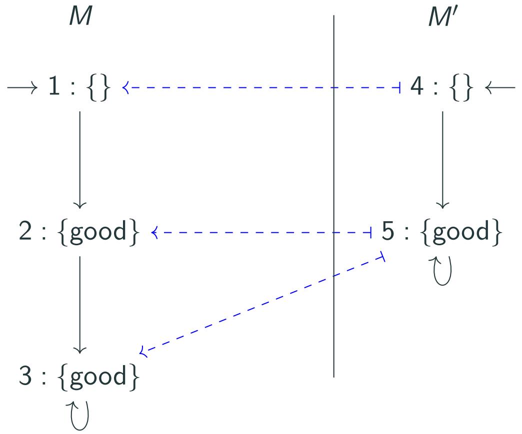
- all paths π from 1 satisfy F A G p, so
- all paths π from 1 eventually reach a state where A G p holds

But path kind (L1) does not: never leaves 1, and in A G p is not satisfied, because there exists a path π_2 that goes to 2 from there.

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

Why have simulation relations and not simulation functions?

$AP = AP' = \{\text{good}\}$



M simulates M'

Compositional model checking?

- "Compositional model checking", E.M. Clarke; D.E. Long; K.L. McMillan (1989)
- "Compositional Model Checking for Multi-Properties", O. Goudsmid, O. Grumberg, S. Sheinvald (2021)

Good luck!