Hoare logic and Model checking

Revision class

Christopher Pulte cp526
University of Cambridge

CST Part II – 2022/23

Hoare logic and separation logic

Structural rules in separation logic

We've used:

- frame rule
  (in proof outlines: indentation)
- rule for existential variables
  (in proof outlines: indentation)
- rule of consequence, as in Hoare Logic
  (in proof outlines: sequence of state assertions)

The concept of ownership

Ownership of a heap cell is the permission to (safely) read/write/dispose of it.

Essential: this ownership is not duplicable.

The concept of ownership (continued)

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

Separation logic:	**If ownership was duplicable:**
$\{X \mapsto v\}$	$\{X \mapsto v\}$
dispose(X);	$\{X \mapsto v * X \mapsto v\}$
$\{\mathit{emp}\}$	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.)

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

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

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

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 $\dots \wedge$ 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:

$\{N = n \land N \geq 0\}$
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 seems to be missing emp in the pre-condition:** $\{N = n \land N \geq 0 \land \text{emp}\}$

Why? $\{N = n \land N \geq 0\}$ 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 to prove the post-condition $\{\text{list}(1, \ldots, n)\}$, 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

$$\cdots \land 1 \leq S$$

instead of

$$\cdots * 1 \leq S$$

.

Conjunction and separating conjunction

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

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

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

Conjunction and separating conjunction (continued)

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

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

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

- $p_1 \mapsto v_1 * p_2 \mapsto v_2$ 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 $p_1 \neq p_2$.
- $p_1 \mapsto v_1 \land p_2 \mapsto v_2$ 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 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.

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

Program variable assignment vs heap assignment

(Program variable) assignment
$X := E$ updates program variable X.

Heap assignment
$[E_1] := E_2$ (note the brackets) evaluates E_1 and, if E_1 evaluates to a pointer to an allocated heap location ℓ, writes to the heap at ℓ.

E.g. heap assignment $[X] := E$ (note the brackets) reads program variable X and, if the current value of X is a pointer to an allocated heap location ℓ, writes to the heap at ℓ, leaving X unchanged.

Whether to apply the rule for **(program variable) assignment** from lecture 1, or the separation logic rule for **heap assignment** depends on the command.

Assignment

Is there a special proof rule for $X := null$? No. This command is a (program variable) assignment, so we would use the (program variable) assignment rule from lecture 1. Separation logic inherits all the partial correctness rules from Hoare logic from the first lecture.

($[X] := null$ would have been a heap assignment.)

Proof for empty list triple?

$\{emp\}$

$\{null = null \wedge emp\}$

$\{[null/X](X = null \wedge emp)\}$

$X := null$

$\{X = null \wedge emp\}$

$\{list(X, [])\}$

Step in lecture 5 proof for allocation

These are all applications of the rule of consequence, using some of the properties of separation logic assertions from lecture 5 (interleaved as comments, in blue).

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

$\wedge$ commutative

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

emp neutral element for $*$

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

$\vdash_{BI} (P \wedge Q) * R \Leftrightarrow P \wedge (Q * R)$ when P is pure

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

$\vdash_{BI} P * Q \Leftrightarrow Q * P$

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

More detailed proof outline for *max*

The *max* operation iterates over a non-empty list, computing its maximum element:

$C_{max} \equiv$

$X := [HEAD + 1]; M := [HEAD];$

$\textbf{while } X \neq \textbf{null do}$

$(E := [X]; (\textbf{if } E > M \textbf{ then } M := E \textbf{ else skip}); X := [X + 1])$

We wish to prove that C_{max} satisfies its intended specification:

$\{list(HEAD, h :: \alpha)\}\ C_{max}\ \{list(HEAD, h :: \alpha) \wedge M = maxl(h :: \alpha)\}$

More detailed proof outline for *max*

$\{list(HEAD, h :: \alpha)\}$

$\{\exists y.\ HEAD \mapsto h, y * list(y, \alpha)\}$

$X := [HEAD + 1];$

$\{\exists y.\ (HEAD \mapsto h, y * list(y, \alpha)) \wedge X = y\}$

$\{HEAD \mapsto h, X * list(X, \alpha)\}$

$M := [HEAD];$

$\{(HEAD \mapsto h, X * list(X, \alpha)) \wedge M = h\}$

$\{(HEAD \mapsto h, X * emp * list(X, \alpha)) \wedge M = h\}$

$\{(plist(HEAD, [h], X) * list(X, \alpha)) \wedge M = h\}$

$\{(plist(HEAD, [h], X) * list(X, \alpha)) \wedge M = maxl([h])\}$

$\{\exists \beta, \gamma.\ h :: \alpha = \beta +\!\!+ \gamma \wedge (plist(HEAD, \beta, X) * list(X, \gamma)) \wedge M = maxl(\beta)\}$

$\textbf{while } X \neq \textbf{null do}$

$(E := [X]; (\textbf{if } E > M \textbf{ then } M := E \textbf{ else skip}); X := [X + 1])$

$\{list(HEAD, h :: \alpha) \wedge M = maxl(h :: \alpha)\}$

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

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

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.

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

$$\begin{tikzcd}
 & {} \arrow[d] & \\
3 : \{\} \arrow[loop above] & 1 : \{\} \arrow[l] \arrow[r] & 2 : \{p\} \arrow[loop above]
\end{tikzcd}$$

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

$$\begin{tikzcd}
{} \arrow[r] & 4 : \{q\} \arrow[r] & 5 : \{p\} \arrow[loop above]
\end{tikzcd}$$

LTL/CTL expressivity

Why are $\mathsf{F\ G}\ p$ in LTL and $\mathsf{A\ F\ A\ G}\ p$ in CTL not equivalent?

$$\begin{tikzcd}
{} \arrow[r] & 1 : \{p\} \arrow[loop below] \arrow[r] & 2 : \{\} \arrow[r] & 3 : \{p\} \arrow[loop below]
\end{tikzcd}$$

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 $\mathsf{F\ G}\ p$ holds.

Informally: $\mathsf{A\ F\ A\ G}\ p$ holds if (check CTL (CTL*) semantics):

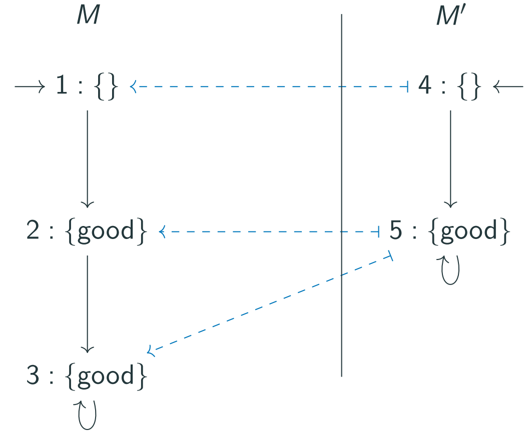
- all paths π from 1 satisfy $\mathsf{F\ A\ G}\ p$, so
- all paths π from 1 eventually reach a state where $\mathsf{A\ G}\ p$ holds

But path kind (L1) does not: never leaves 1, and in 1, $\mathsf{A\ G}\ p$ 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'

Good luck!