I

Variables as Resource in Hoare Logics

Matthew Parkinson and Richard Bornat
School of Computing
Middlesex University
LONDON, UK
mjp41@cam.ac.uk, R.Bornat@mdx.ac.uk

Cristiano Calcagno
Department of Computing
Imperial College
University of London, LONDON, UK
ccris@doc.ic.ac.uk

Abstract

Hoare logic is bedevilled by complex but coarse side conditions on the use of variables. We define a logic, free of side conditions, which permits more precise statements of a program's use of variables. We show that it admits translations of proofs in Hoare logic, thereby showing that nothing is lost, and also that it admits proofs of some programs outside the scope of Hoare logic. We include a treatment of reference parameters and global variables in procedure call (though not of parameter aliasing). Our work draws on ideas from separation logic: program variables are treated as resource rather than as logical variables in disguise. For clarity we exclude a treatment of the heap.

Following [3], a total permission $\top$ may be split into two read permissions, which may themselves be split further, and split permissions may be recombined ($p \circledast p'$). Any permission at all gives read access.

There is a set of permissions Perms, equipped with a partial function $\circledast$: Perms $\times$ Perms $\rightharpoonup$ Perms and a distinguished element $\top \in$ Perms, such that (Perms, $\circledast$) forms a partial cancellative[1] commutative semigroup with the properties divisibility, total permission, and no unit:

$$
\begin{aligned}
\forall c \in \mathsf{Perms} \cdot \exists c', c'' \in \mathsf{Perms} \cdot (c' \circledast c'' = c) \\
\forall c \in \mathsf{Perms} \cdot (\top \circledast c \text{ is undefined}) \\
\forall c, c' \in \mathsf{Perms} \cdot (c \circledast c' \neq c)
\end{aligned}
$$

Example models are: (1) Perms $= \{z \mid 0 < z \leq 1\}$, $\top = 1$, $\circledast$ is $+$ (only defined if the result does not exceed 1); (2) Perms $= \{S \mid S \subseteq \mathbb{N}, S \text{ infinite}\}$, $\top = \mathbb{N}$, $\circledast$ is $\uplus$.

ι ranges over elements of Perms. Permission expressions π have the following syntax:

$$\pi ::= \iota \mid p \mid \pi \circledast \pi$$

Stacks s are finite partial maps from program variable names to pairs of an integer and a permission. Interpretations i are finite partial maps from logical variable names to integers and permissions. We only consider interpretations that define all the logical variables we use.

$$
\begin{aligned}
s : \mathcal{S} &\stackrel{\text{def}}{=} \mathsf{PVarNames} \rightharpoonup_{\text{fin}} \mathsf{Int} \times \mathsf{Perms} \\
i : \mathcal{R} &\stackrel{\text{def}}{=} \mathsf{LVarNames} \rightharpoonup_{\text{fin}} \mathsf{Int} \cup \mathsf{Perms}
\end{aligned}
$$

We use $[\![E]\!]_{(s,i)}$ for the (partial) evaluation of expressions, and $[\![E]\!]_s$ will do when E does not contain logical variables:

$$
\begin{aligned}
[\![E1 + E2]\!]_{(s,i)} &= [\![E1]\!]_{(s,i)} + [\![E2]\!]_{(s,i)} \\
[\![0]\!]_{(s,i)} &= 0 \\
[\![x]\!]_{(s,i)} &= \begin{cases} (s(x))_1 & x \in dom(s) \\ \text{undefined} & \text{otherwise} \end{cases} \\
[\![X]\!]_{(s,i)} &= i(X)
\end{aligned}
$$

We define a (partial) evaluation operation on permissions expressions:

$$
\begin{aligned}
[\![\pi 1 \circledast \pi 2]\!]_{(s,i)} &= [\![\pi 1]\!]_{(s,i)} \circledast [\![\pi 2]\!]_{(s,i)} \\
[\![\iota]\!]_{(s,i)} &= \iota \\
[\![p]\!]_{(s,i)} &= i(p)
\end{aligned}
$$

Table 1. Forcing Semantics $(s,i) \models \Phi$

$$
\begin{aligned}
(s,i) &\models E1 = E2 &&\iff [\![E1]\!]_{(s,i)} = [\![E2]\!]_{(s,i)} \wedge [\![E1]\!]_{(s,i)} \text{ and } [\![E2]\!]_{(s,i)} \text{ are defined}\\
(s,i) &\models \Phi \Rightarrow \Phi' &&\iff ((s,i) \models \Phi) \Rightarrow ((s,i) \models \Phi')\\
(s,i) &\models \Phi \star \Phi' &&\iff \exists s1, s2 \cdot (s = s1 \star s2 \wedge ((s1,i) \models \Phi) \wedge ((s2,i) \models \Phi'))\\
(s,i) &\models \Phi \mathrel{-\!\!\star} \Phi' &&\iff \forall s1 \cdot (s\#s1 \wedge ((s1,i) \models \Phi) \Rightarrow ((s \star s1, i) \models \Phi'))\\
(s,i) &\models \mathrm{Own}_\pi(x) &&\iff [\![\pi]\!]_{(s,i)} \text{is defined} \wedge s = \{\langle x, (\_, [\![\pi]\!]_{(s,i)})\rangle\}\\
(s,i) &\models \mathbf{emp_s} &&\iff s = \{\}\\
(s,i) &\models \text{false} &&\iff \text{false}\\
(s,i) &\models \forall X \cdot \Phi &&\iff \forall v \cdot ((s, i \oplus \langle X, v\rangle) \models \Phi)
\end{aligned}
$$

We encode true, $\wedge$, $\vee$, $\exists$ and $\neg$: e.g. $A \vee B$ is $(A \Rightarrow \text{false}) \Rightarrow B$.

$$s\#s' \iff \forall x, v, v', \iota, \iota' \cdot (s(x) = (v,\iota) \wedge s'(x) = (v', \iota') \Rightarrow v = v' \wedge \exists \iota'' \cdot (\iota'' = \iota \circledast \iota'))$$

$$
s \star s' = \begin{cases}
\left\{ \langle x, (v,\iota)\rangle \;\middle|\; \begin{array}{l} (s(x) = (v,\iota) \wedge x \notin \mathit{dom}(s')) \\ \vee (s'(x) = (v,\iota) \wedge x \notin \mathit{dom}(s)) \\ \vee (s(x) = (v,\iota') \wedge s'(x) = (v, \iota'') \wedge \iota = \iota' \circledast \iota'') \end{array} \right\}, & \text{where } s\#s';\\
\text{undefined}, & \text{otherwise.}
\end{cases}
$$

A forcing semantics is given in table 1. $s\#s'$ asserts that two stacks are compatible, agreeing about values where their domains intersect and not claiming too much permission; $s \star s'$ expresses separation of stacks; $\langle a, b\rangle$ is an element of a function; $\oplus$ is function update; $\uplus$ is disjoint function extension.

$\mathrm{Own}_\pi(x)$ asserts ownership of a stack containing a variable called x and permission π to access it. Crucially it also asserts that this is *all* that the stack contains. It says nothing about the content of the variable; it is purely about the lvalue of x (contrast $E \mapsto F$ in separation logic, which asserts a single-cell heap and describes its content). $\mathrm{Own}_\top(x)$ asserts total permission, i.e. ownership, and $\mathrm{Own}_\_(x)$ means $\exists p \cdot (\mathrm{Own}_p(x))$. $\mathbf{emp_s}$ asserts the empty stack, and true holds of any stack at all. Following separation logic, $(\star)$ combines stack assertions: $\mathrm{Own}_\_(x) \star \mathrm{Own}_\_(y)$ is a two-variable stack; $\mathrm{Own}_\pi(x) \star \mathrm{Own}_{\pi'}(x)$ is equivalent to $\mathrm{Own}_{\pi \circledast \pi'}(x)$ and therefore $\mathrm{Own}_\top(x) \star \mathrm{Own}_\pi(x)$ is false; $\mathrm{Own}_\_(x) \star \text{true}$ is a stack which contains at least the variable x.

Arithmetic equality and inequality imply a level of ownership but are *loose* about the stack in which they operate: $x = 1$, for example, implicitly asserts $\mathrm{Own}_\_(x) \star \text{true}$. Our logic does not admit as a tautology $E \neq F \iff \neg(E = F)$. $x \neq 1$, for example, is satisfied by any stack in which there is a cell called x which does not contain 1; $\neg(x = 1)$, on the other hand, is satisfied by the same stacks and by those (for example $\mathbf{emp_s}$) in which x does not occur at all.

Definition 1.

$$x1_{\pi 1}, \ldots, xn_{\pi n} \Vdash P \stackrel{\mathit{def}}{=} (\mathrm{Own}_{\pi 1}(x1) \star \ldots \star \mathrm{Own}_{\pi n}(xn)) \wedge P$$

3.2. Rules

Our programming language is the language of Hoare logic plus variable declarations 'local-in-end' and procedure declarations 'let-=-in-end'. For simplicity we consider procedures each of which have a single call-by-reference parameter x and a single call-by-value parameter y. It would be straightforward to extend this treatment to deal with other cases.

Table 2. Axioms and Rules $\Gamma \vdash_{vr} \{\Phi\}\ C\ \{\Phi\}$

$$\Gamma \vdash_{vr} \{x_\top, O \Vdash X = E\}\ x := E\ \{x_\top, O \Vdash x = X\}$$

$$\Gamma \vdash_{vr} \{\exists X \cdot X = E \wedge (\mathrm{Own}_\top(x) \star ((x = X \wedge \mathrm{Own}_\top(x)) \mathrel{-\!\!\star} \Phi))\}\ x := E\ \{\Phi\} \quad (X \text{ fresh for } \Phi)$$

$$\frac{\Phi \Rightarrow B = B \quad \Gamma \vdash_{vr} \{\Phi \wedge B\}\ C1\ \{\Phi'\} \quad \Gamma \vdash_{vr} \{\Phi \wedge \neg B\}\ C2\ \{\Phi'\}}{\Gamma \vdash_{vr} \{\Phi\}\ \text{if } B \text{ then } C1 \text{ else } C2 \text{ fi}\ \{\Phi'\}} \qquad \frac{\Phi \Rightarrow B = B \quad \Gamma \vdash_{vr} \{\Phi \wedge B\}\ C\ \{\Phi\}}{\Gamma \vdash_{vr} \{\Phi\}\ \text{while } B \text{ do } C \text{ od}\ \{\Phi \wedge \neg B\}}$$

$$\frac{\Gamma \vdash_{vr} \{\mathrm{Own}_\top(z) \star \Phi\}\ C[z/x]\ \{\mathrm{Own}_\top(z) \star \Phi'\}}{\Gamma \vdash_{vr} \{\Phi\}\ \text{local } x \text{ in } C \text{ end}\ \{\Phi'\}} \ (\text{fresh } z)$$

$$\frac{\Phi \Rightarrow \Phi' \quad \Gamma \vdash_{vr} \{\Phi'\}\ C\ \{\Psi'\} \quad \Psi' \Rightarrow \Psi}{\Gamma \vdash_{vr} \{\Phi\}\ C\ \{\Psi\}} \qquad \frac{\Gamma \vdash_{vr} \{\Phi\}\ C\ \{\Psi\}}{\Gamma \vdash_{vr} \{\exists X \cdot \Phi\}\ C\ \{\exists X \cdot \Psi\}} \qquad \frac{\Gamma \vdash_{vr} \{\Phi\}\ C\ \{\Phi'\}}{\Gamma \vdash_{vr} \{\Phi \star \Psi\}\ C\ \{\Phi' \star \Psi\}}$$

$$\frac{\Gamma' \vdash_{vr} \{\Phi\}\ C1\ \{\Phi'\} \quad \Gamma' \vdash_{vr} (\{\Psi \star \mathrm{Own}_\top(y) \wedge y = Y\}\ C\ \{\Psi' \star \mathrm{Own}_\top(y)\})[w, z/x, y]}{\Gamma \vdash_{vr} \{\Phi\}\ \text{let } f(x; y) = C \text{ in } C1 \text{ end}\ \{\Phi'\}} \ (\text{fresh } w, z;\ \Gamma' = \Gamma, \{\Psi\}\, f(x;Y)\, \{\Psi'\})$$

$$\frac{\Gamma, (\{\Psi 1\}\, f(x;Y)\, \{\Psi 2\})[z/x] \vdash_{vr} \{\Phi\}\ C\ \{\Phi'\}}{\Gamma, \{\Psi 1\}\, f(x;Y)\, \{\Psi 2\} \vdash_{vr} \{\Phi\}\ C\ \{\Phi'\}} \ (z \text{ fresh for } \{\Psi 1\}\, f(x;Y)\, \{\Psi 2\})$$

$$\Gamma \vdash_{vr} \{(\Phi \star \Psi) \wedge Y = E\}\ f(x;E)\ \{\Phi' \star \Psi\} \quad (\{\Phi\}\, f(x;Y)\, \{\Phi'\} \in \Gamma)$$

The rules of our program logic are given in table 2.[2] Γ is the function context, a set of specifications $\{\Phi\}f(x;Y)\{\Phi'\}$, and O ranges over ownership assertions $x1_{\pi 1}, \ldots, xn_{\pi n}$. The first assignment axiom can be used in forward reasoning. The second is a weakest pre-condition version which can be derived from the first. The if and while rules have an antecedent $\Phi \Rightarrow B{=}B$, which ensures that variables mentioned in B are in the stack. In the let rule we give the function body C total permission to access the value parameter y. The first function-call rule deals with reference arguments by straightforward α-conversion. The second, an axiom, deals with value arguments, and is subtle. You might have expected to see

$$\Gamma, \{\Phi\}\, f(x;Y)\, \{\Phi'\} \vdash_{vr} \{\Phi[E/Y]\}\ f(x;E)\ \{\Phi'[E/Y]\}$$

But suppose that Φ is $Y = 3 \wedge \mathbf{emp_s}$: then Φ claims no stack, but $\Phi[E/Y]$ is $E = 3 \wedge \mathbf{emp_s}$, which is false if E mentions any program variables. Or you you might have expected

$$\Gamma, \{\Phi\}\, f(x;Y)\, \{\Phi'\} \vdash_{vr} \{\Phi \wedge Y = E\}\ f(x;E)\ \{\Phi'\}$$

But if Φ is $Y = 3 \wedge \mathrm{Own}_\top(x)$, then the precondition $Y = 3 \wedge \mathrm{Own}_\top(x) \wedge Y = E$ is false if E mentions any program variables other than x. In the axiom of table 2 Ψ claims the stack that E claims but Φ does not, and $(\Phi \star \Psi) \wedge Y = E$ allows the procedure call to read and/or write variables that are mentioned both in E and Φ as well as to be provided with a value to use in place of Y.

Table 3. Operational semantics $s \xrightarrow{C}{}^{n}_{\rho}\, s'$ **and** $s \xrightarrow{C}{}^{n}_{\rho}$ unsafe

$$s \xrightarrow{\mathsf{skip}}{}^{n}_{\rho}\, s \qquad\qquad \frac{s(x) = (\_, \top)}{s \xrightarrow{x:=E}{}^{n}_{\rho}\, s \oplus \langle x, ([\![E]\!]_{s}, \top)\rangle}$$

$$\frac{[\![B]\!]_{s} = \text{true} \quad s \xrightarrow{C_{\mathsf{true}}}{}^{n}_{\rho}\, s'}{s \xrightarrow{\mathsf{if}\ B\ \mathsf{then}\ C_{\mathsf{true}}\ \mathsf{else}\ C_{\mathsf{false}}\ \mathsf{fi}}{}^{n}_{\rho}\, s'} \qquad \frac{[\![B]\!]_{s} = \text{false} \quad s \xrightarrow{C_{\mathsf{false}}}{}^{n}_{\rho}\, s'}{s \xrightarrow{\mathsf{if}\ B\ \mathsf{then}\ C_{\mathsf{true}}\ \mathsf{else}\ C_{\mathsf{false}}\ \mathsf{fi}}{}^{n}_{\rho}\, s'}$$

$$\frac{s \xrightarrow{\mathsf{if}\ B\ \mathsf{then}\ (C; \mathsf{while}\ B\ \mathsf{do}\ C\ \mathsf{od})\ \mathsf{else\ skip\ fi}}{}^{n}_{\rho}\, s'}{s \xrightarrow{\mathsf{while}\ B\ \mathsf{do}\ C\ \mathsf{od}}{}^{n}_{\rho}\, s'} \qquad \frac{s \xrightarrow{C1}{}^{n}_{\rho}\, s' \quad s' \xrightarrow{C2}{}^{n}_{\rho}\, s''}{s \xrightarrow{C1;C2}{}^{n}_{\rho}\, s''}$$

n+1 ?

$$\frac{s \uplus \langle z, (\_, \top)\rangle \xrightarrow{C[z/x]}{}^{n}_{\rho}\, s' \uplus \langle z, (\_, \top)\rangle}{s \xrightarrow{\mathsf{local}\ x\ \mathsf{in}\ C\ \mathsf{end}}{}^{n}_{\rho}\, s'} \ (\text{fresh } z) \qquad \frac{s \xrightarrow{C'}{}^{n}_{\rho \oplus \langle f, (y,z,C)\rangle}\, s'}{s \xrightarrow{\mathsf{let}\ f(y;z)=C\ \mathsf{in}\ C'\ \mathsf{end}}{}^{n}_{\rho}\, s'}$$

$$\frac{\rho(f) = (y, z, C) \quad s \xrightarrow{\mathsf{local}\ z\ \mathsf{in}\ z:=E; C[x/y]\ \mathsf{end}}{}^{n}_{\rho}\, s'}{s \xrightarrow{f(x;E)}{}^{n+1}_{\rho}\, s'} \ (\text{fresh } z')$$

?

$$\frac{\langle x, (\_, \top)\rangle \notin s}{s \xrightarrow{x:=E}{}^{n}_{\rho} \text{ unsafe}} \qquad \frac{[\![E]\!]_{s} \text{ is undefined}}{s \xrightarrow{x:=E}{}^{n}_{\rho} \text{ unsafe}} \qquad \frac{[\![B]\!]_{s} \text{ is undefined}}{s \xrightarrow{\mathsf{if}\ B\ \mathsf{then}\ C1\ \mathsf{else}\ C2\ \mathsf{fi}}{}^{n}_{\rho} \text{ unsafe}}$$

* Note that logical variable are not allowed here.

3.3. Soundness

An operational semantics is given in table 3. In $s \xrightarrow{C}{}^{n}_{\rho}\, s'$

- s and s' are stacks;
- C is a command;
- ρ maps procedure names to a triple (x, y, C') of reference-parameter name x, value-parameter name y and command C'; and
- n is a recursion-depth counter.

A *safe computation* – the top part of the table and definition 2 – does not access stack locations that are undefined. The lower part of the table deals with unsafe computations, which access variables for which they have no permission.

Definition 2. $s \xrightarrow{C}{}^{n}_{\rho}$ safe *iff* $\forall n. \neg(s \xrightarrow{C}{}^{n}_{\rho}$ unsafe$)$

?

Lemma 3. *If* $s \xrightarrow{C}{}^{n}_{\rho}$ safe *and* $s' \# s$ *then* $s \star s' \xrightarrow{C}{}^{n}_{\rho}$ safe

Proof. By induction on the evaluation rules. □

Lemma 4 (Locality). *If* $s \xrightarrow{C}{}^{n}_{\rho}$ safe *and* $s' \# s$ *and* $s \star s' \xrightarrow{C}{}^{n}_{\rho}\, s1$ *then* $\exists s2 \cdot s \xrightarrow{C}{}^{n}_{\rho}\, s2$ *and* $s2 \star s' = s1$.

Proof. By induction on the evaluation rules. □

Choice of fresh variable does not affect the reduction, and hence the semantics are deterministic with respect to the stack.

Definition 5 (Variable interchange: $\leftrightarrow$).

$$((y \leftrightarrow x)s)\ x \stackrel{def}{=} ((x \leftrightarrow y)s)\ x \stackrel{def}{=} s\ y;$$
$$((x \leftrightarrow y)s)\ z \stackrel{def}{=} s\ z.$$

Lemma 6.

$$(z \leftrightarrow x)s \xrightarrow{C[z/x]}{}^{n}_{\rho} (z \leftrightarrow x)s' \Rightarrow s \xrightarrow{C}{}^{n}_{\rho} s'$$
$$(z \leftrightarrow x)s \xrightarrow{C[z/x]}{}^{n}_{\rho} \mathsf{unsafe} \iff s \xrightarrow{C}{}^{n}_{\rho} \mathsf{unsafe},$$
$$(z \textit{ fresh for } C \textit{ and } \rho,\ x \notin \mathrm{dom}(s)).$$

Proof. By induction on the evaluation rules. □

Lemma 7 (Determinacy).
If $s \xrightarrow{C}{}^{n}_{\rho} s1$ *and* $s \xrightarrow{C}{}^{n}_{\rho} s2$ *then* $s1 = s2$.

Proof. By induction on the evaluation rules. The rules for local require lemma 6. Other rules hold trivially. □

In the semantics of triples, the precondition implies a safe computation, in contrast to the semantics of standard Hoare logic.

Definition 8.

$$\rho \vDash_n \{\Phi\}C\{\Phi'\} \stackrel{def}{=} \forall s, s', i \cdot \left((s,i) \vDash \Phi \Rightarrow \left(\begin{array}{l} s \xrightarrow{C}{}^{n}_{\rho} \mathsf{safe} \wedge \\ (s \xrightarrow{C}{}^{n}_{\rho} s' \Rightarrow (s',i) \vDash \Phi') \end{array} \right) \right)$$

Definition 9.

$\rho \vDash_n \Gamma \stackrel{def}{=}$ *for every* $\{\Phi\}\ f(x;Y)\ \{\Phi'\}$ *in* Γ, $\langle f, (x', y, C)\rangle$ *is in* ρ *such that, for fresh* z *and* w,

$$\rho \vDash_n \left(\begin{array}{c} \{\Phi \star (y_\top \Vdash y = X)\} \\ C[x/x'] \\ \{\Phi' \star \mathrm{Own}_\top(y)\} \end{array} \right) [z, w/x, y]$$

Definition 10 (Semantics of judgements).

$$\Gamma \vDash_n \{\Phi\}C\{\Phi'\} \stackrel{def}{=}$$
$$\forall \rho \cdot (\ (\rho \vDash_n \Gamma) \Rightarrow (\rho \vDash_{n+1} \{\Phi\}C\{\Phi'\})\ )$$

Theorem 11. *If* $\Gamma \vdash_{vr} \{\Phi\}\ C\ \{\Phi'\}$ *is derivable then* $\forall n \cdot (\Gamma \vDash_n \{\Phi\}C\{\Phi'\})$

Proof. By induction on the derivation. □

4. Substitution

In Hoare logic substitution is used to model assignment and parameter passing, but simple properties of substitution do not hold in our logic. In particular, substitution of formulae can affect ownership. $X = E \wedge \Phi \Rightarrow \Phi[E/X]$, for example, is not a tautology. (Here is a counter-example:

$$\begin{aligned} &X = E \wedge ((X = X \wedge \mathbf{emp_s}) \star E = E) \\ &\not\Rightarrow (E = E \wedge \mathbf{emp_s}) \star E = E) \end{aligned}$$

– the left side of the implication is satisfiable, while the right is false if E contains program variables.) In the rest of this section we consider a subset of the logic in which substitution is well-behaved. As a result, we derive the following assignment axiom.

$$\Gamma \vdash_{vr} \{x_\top, O \Vdash \phi[E/x] \wedge E = E\}\ x := E\ \{x_\top, O \Vdash \phi\} \tag{2}$$

A stack-imprecise formula does not notice extension of the stack and does not care about the quantity of permission it has for any variable.

Definition 12. Φ *is* stack imprecise $\stackrel{def}{=}$
$\forall s, s', i \cdot$
$\quad (((s,i) \vDash \Phi) \wedge \lfloor s \rfloor \subseteq \lfloor s' \rfloor \Rightarrow ((s',i) \vDash \Phi))$
where $\lfloor s \rfloor = \{\langle x, v\rangle \mid \langle x, (v,p)\rangle \in s\}$

Lemma If $s = s_1 * s_2$ then $\lfloor s_1 \rfloor \subseteq \lfloor s \rfloor$.

Definition $s_{1/2} = \{ < x, (v, \frac{1}{2}p) > \mid < x, (v, p) > \in s \}$.

Lemma $\lfloor s_{\frac{1}{2}} \rfloor = \lfloor s \rfloor$ and $s_{\frac{1}{2}} \# s_{1/2}$ and $s = s_{1/2} * s_{1/2}$.

Lemma 13. *If Φ and Ψ are stack imprecise, then*
$\models \Phi \star \Psi \;\Leftrightarrow\; \Phi \wedge \Psi$

Proof Suppose $s, i \models \Phi * \Psi$. Then there are s_1, s_2 such that

$s_1 \# s_2$ and $s = s_1 * s_2$ and $s_1, i \models \Phi$ and $s_2, i \models \Psi$.

But $\lfloor s_1 \rfloor \subseteq \lfloor s \rfloor$ and $\lfloor s_2 \rfloor \subseteq \lfloor s \rfloor$, so that

$(s, i) \models \Phi$ and $(s, i) \models \Psi$,

so that $(s, i) \models \Phi \wedge \Psi$.

Suppose $(s, i) \models \Phi \wedge \Psi$. Then

$(s, i) \models \Phi$ and $s, i \models \Psi$.

But $\lfloor s_{1/2} \rfloor = \lfloor s \rfloor$, so that

$(s_{1/2}, i) \models \Phi$ and $(s_{1/2}, i) \models \Psi$.

Also, $s_{1/2} * s_{1/2} = s$, so that $(s, i) \models \Phi * \Psi$.

Lemma $E = E'$ is stack-imprecise.

Corollary 14. *If Φ is stack imprecise, then*
$\models \Phi \star E = E' \;\Leftrightarrow\; \Phi \wedge E = E'$

We define implication in the same way as when intuitionistic implication is encoded into classical separation logic [12].

Definition 15 (Stack-imprecise $\Rightarrow$ and $\neg$).
$\Phi \overset{s}{\Rightarrow} \Phi' \overset{def}{=} \text{true} \mathrel{-\!\!*} (\Phi \Rightarrow \Phi')$ *and* $\overset{s}{\neg}\Phi \overset{def}{=} \Phi \overset{s}{\Rightarrow} \text{false}$.

Note: $E \neq E' \iff \overset{s}{\neg}(E = E')$ is a tautology.

Proof.

(A) $(s,i) \models E \neq E'$ iff
iff $[\![E]\!]_{s,i}$ and $[\![E']\!]_{s,i}$ defined
and $[\![E]\!]_{s,i} \neq [\![E']\!]_{s,i}$.

$(s,i) \models \overset{s}{\neg} E = E'$
iff $(s,i) \models E = E' \overset{s}{\Rightarrow}$ false
iff $(s,i) \models \text{true} \mathrel{-\!\!*} (E = E' \Rightarrow \text{false})$
iff $\forall s_1.\ s \# s_1$ and $(s_1, i) \models$ true implies
$(s * s_1, i) \models (E = E' \Rightarrow \text{false})$
iff $\forall s_1.\ \underbrace{s \# s_1 \text{ implies not } (s * s_1, i) \models E = E'}_{(B)}$

	(A)	(B)	$\forall s.$(B)
$[\![E]\!]_{s,i}$ or $[\![E']\!]_{s,i}$ undefined	false	true false ?	false
$[\![E]\!]_{s,i}$ and $[\![E']\!]_{s,i}$ defined and $[\![E]\!]_{s,i} = [\![E']\!]_{s,i}$	false	false	false
$[\![E]\!]_{s,i}$ and $[\![E']\!]_{s,i}$ defined and $[\![E]\!]_{s,i} \neq [\![E']\!]_{s,i}$	~~false~~ true	true	true

Oops

In the preceeding proof sketch, we assumed that, if $[\![E]\!]_{s,i}$ or $[\![E']\!]_{s,i}$ is undefined, then there will be some s_1 such that $s \# s_1$ and $(s * s_1, i) \models E = E'$ is true, so that

$$s \# s_1 \text{ implies not } (s * s_1, i) \models E = E' \qquad \text{(B)}$$

is false.

But $(s * s_1, i) \models x = x+1$ is never true, so that (B) is never false — and the proof fails.

In fact, when x is undefined, $x \neq x+1$ is false but $\overset{s}{\neg}(x = x+1)$ is true. Thus

$$E \neq E' \Longleftrightarrow \overset{s}{\neg}(E = E')$$

is not a tautology.

Definition 16 (restricted formulae).
$\phi ::= E{=}E \mid \phi \wedge \phi \mid \phi \vee \phi \mid \phi \overset{s}{\Rightarrow} \phi \mid \phi \mathrel{-\!\!\star} \phi \mid \pi = \pi \mid$
$\phi \star \phi \mid \forall X.\phi \mid \exists X.\phi \mid \mathrm{false} \mid \mathrm{true} \mid \overset{s}{\neg} \phi$

What's missing: emp, $\mathrm{Own}_\pi(x)$, classical $\Rightarrow$ and $\neg$.

Lemma 17. *Restricted formulae are stack imprecise.*

Proof. Structural induction on ϕ. □

Lemma 18.
$(s,i) \vDash X = E \quad \Rightarrow \quad [\![E']\!]_{(s,i)} = [\![E'[E/X]]\!]_{(s,i)}$

Proof. By induction on structure of E' □

Lemma 19. $\vDash X = E \quad \Rightarrow \quad (\phi \Leftrightarrow \phi[E/X])$

Proof. By structural induction on ϕ. The ($\star$) and ($\mathrel{-\!\!\star}$) cases require lemma 14, and the ($=$) case requires lemma 18. □

Definition 20. $\mathrm{vars}(O) \overset{def}{=} \{x \mid (x)_p \in O\}$

Suppose $O = x^1_{\pi^1}, \ldots, x^n_{\pi^n}$. We define

$$\langle O \rangle \equiv \mathrm{Own}_{\pi^1}(x^1) * \cdots * \mathrm{Own}_{\pi^n}(x^n)$$

so that

$$O \Vdash P = \langle O \rangle \wedge P \qquad O \Vdash \mathrm{true} = \langle O \rangle.$$

Lemma 21. $(O1 \Vdash \phi 1) \star (O2 \Vdash \phi 2) \Rightarrow (O1, O2 \Vdash \phi 1 \star \cancel{\phi 1})$
$\phi 2$

Proof

$$(p_1 \wedge p_2) * (q_1 \wedge q_2) \Rightarrow (p_1 * (q_1 \wedge q_2)) \wedge (p_2 * (q_1 \wedge q_2))$$
$$\Rightarrow (p_1 * q_1) \wedge (p_1 * q_2) \wedge (p_2 * q_1) \wedge (p_2 * q_2)$$
$$\Rightarrow (p_1 * q_1) \wedge (p_2 * q_2)$$

Then

$$(O1 \Vdash \phi 1) * (O2 \Vdash \phi 2) \Leftrightarrow (\langle O1 \rangle \wedge \phi 1) * (\langle O2 \rangle \wedge \phi 2)$$
$$\Rightarrow (\langle O1 \rangle * \langle O2 \rangle) \wedge (\phi 1 * \phi 2) \Leftrightarrow O1, O2 \Vdash \phi 1 * \phi 2$$

Lemma 22. *If* $\mathrm{FV}(\phi 1) \subseteq \mathrm{vars}(O1)$ *and* $\mathrm{FV}(\phi 2) \subseteq \mathrm{vars}(O2)$ *and* $\models O \Vdash \mathrm{true} \iff O1 \Vdash \mathrm{true} \star O2 \Vdash \mathrm{true}$ *then* $\models (O \Vdash \phi 1 \star \phi 2) \Rightarrow (O1 \Vdash \phi 1) \star (O2 \Vdash \phi 2)$.

Proof Suppose $s, i \models \langle O \rangle \wedge (\phi 1 * \phi 2)$. Then there are s_1, s_2 such that

$$s = s_1 * s_2 \text{ and } s_1, i \models \phi 1 \text{ and } s_2, i \models \phi 2.$$

Moreover, $s, i \models \langle O \rangle$, and since $\langle O \rangle \Longleftrightarrow \langle O1 \rangle * \langle O2 \rangle$, there are s_1', s_2' such that

$$s = s_1' * s_2' \text{ and } s_1', i \models \langle O1 \rangle \text{ and } s_2' \models \langle O2 \rangle.$$

Since $s_1, i \models \phi 1$, we have $\mathrm{FV}(\phi 1) \subseteq \mathrm{dom}\, s_1$ and

$$s_1 \upharpoonright \mathrm{FV}(\phi 1), i \models \phi 1.$$

Since $s_1', i \models \langle O1 \rangle$, we have

$$\mathrm{vars}(O1) \subseteq \mathrm{dom}\, s_1'.$$

Since $s = s_1 * s_2 = s_1' * s_2'$,

$\lfloor s_1 \rfloor$ and $\lfloor s_1' \rfloor$ are both restrictions of $\lfloor s \rfloor$,

and since

$$\mathrm{dom}(\lfloor s_1 \rfloor \upharpoonright \mathrm{FV}(\phi 1)) = \mathrm{dom}(s_1 \upharpoonright \mathrm{FV}(\phi 1)) = \mathrm{FV}(\phi 1)$$
$$\subseteq \mathrm{vars}(O1) \subseteq \mathrm{dom}\, s_1' = \mathrm{dom} \lfloor s_1' \rfloor,$$

we have $\lfloor s_1 \rfloor \upharpoonright \mathrm{FV}(\phi 1) \subseteq \lfloor s_1' \rfloor$.

Then, since $s_1 \upharpoonright \mathrm{FV}(\phi 1), i \models \phi 1$ and $\phi 1$ is stack-imprecise,

$$s_1', i \models \langle O1 \rangle \wedge \phi 1.$$

Similarly, $s_2', i \models \langle O2 \rangle \wedge \phi 2$. Then, since $s = s_1' * s_2'$,

$$s, i \models (\langle O1 \rangle \wedge \phi 1) * (\langle O2 \rangle \wedge \phi 2).$$

Table 6. Variables-as-resource rules for concurrency

$$\frac{\Delta \vdash_{vr} \{\Phi1\}\ \mathit{C1}\ \{\Phi1'\} \quad \Delta \vdash_{vr} \{\Phi2\}\ \mathit{C2}\ \{\Phi2'\}}{\Delta \vdash_{vr} \{\Phi1 \star \Phi2\}\ \mathit{C1} \| \mathit{C2}\ \{\Phi1' \star \Phi2'\}} \qquad \frac{\Delta, b : \Psi \vdash_{vr} \{\Phi\}\ C\ \{\Phi'\}}{\Delta \vdash_{vr} \{\Phi \star \Psi\}\ \text{resource } b \text{ in } C \text{ end }\ \{\Phi' \star \Psi\}}$$

$$\frac{\Delta \vdash_{vr} \{(\Phi \star \Psi) \wedge B\}\ C\ \{\Phi' \star \Psi\} \quad \Phi \star \Psi \Rightarrow B = B}{\Delta, b : \Psi \vdash_{vr} \{\Phi\}\ \text{with } b \text{ when } B \text{ do } C \text{ od }\ \{\Phi'\}}$$