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.

1. Introduction

The glory of Hoare logic [10] is the variable-assignment axiom, which converts difficult semantic arguments about program state into simple syntactic substitutions. That success depends on punning program variables in commands with identically-named logical variables in assertions, but program variables are not logical variables: they have location (lvalue or lv in Strachey's classification [19], otherwise 'address') as well as value (Strachey's rvalue or rv, otherwise 'content').

The price of the pun is a proliferation of well-chosen but complex side conditions on the invariance rule, on procedure-call [11, 9, 7] and on concurrency [16]. The invariance rule, for example, is

$$\frac{\{Q\}\,C\,\{R\}}{\{Q \wedge P\}\,C\,\{R \wedge P\}} \ \mathrm{mods}(C) \cap \mathrm{FV}(P) = \varnothing$$

The side condition lets the programmer know that variables not in $\mathrm{mods}(C)$ are preserved. We can suppose that this imposes a useful discipline, that the writing footprint of commands should be considered when setting them in a wider

context. But the side condition is too coarse. Consider, for example the procedure definition

$$\text{let } f(b) = \text{if } b \text{ then } x := 1 \text{ else } y := 0 \text{ fi}$$

It is obvious that $f(true)$ does not modify y and $f(false)$ does not modify x, but a simple modifies clause would have it that $\mathrm{mods}(f(\_))$ is $\{x, y\}$, and the invariance rule will not help us to establish

$$\{y = 3\}\, f(\text{true})\, \{x = 1 \wedge y = 3\} \tag{1}$$

Side conditions on the use of global variables in procedures, on reference parameters and on concurrency are equally coarse but far more complicated, and can call for global oversight to establish the validity of a small part of a program; the effect can be that local changes in variable use can invalidate already-established proofs of distant parts of a program.

In the logic we present in this paper we can make more precise statements about the variable-resource footprints of program parts. We can specify, for example

$$
\begin{aligned}
&\{(x_\top \Vdash B) \vee (y_\top \Vdash \neg B)\} \\
&\quad f(B) \\
&\{(x_\top \Vdash x = 1 \wedge B) \vee (y_\top \Vdash y = 0 \wedge \neg B)\}
\end{aligned}
$$

– $f(\text{true})$ has total permission to read and write x, but cannot touch y, and vice-versa $f(\text{false})$ can read and write y but cannot access x. We can then use separation logic's frame rule (see table 2), a version of the invariance rule, to establish

$$\{x_\top, y_\_ \Vdash y = 3\}\, f(\text{true})\, \{x_\top, y_\_ \Vdash x = 1 \wedge y = 3\}$$

This is a more precise statement of (1), stating that a program which has permission to write x ($x_\top$) and read y ($y_\_$) can call $f(\text{true})$ and rely on the fact that the value of y will not be changed.

By requiring variable-resource descriptions in assertions, we have pushed the side conditions into the logic, eliminating them from the rules. This does not eliminate the need

for the programmer to follow rules of program hygiene, but it does allow the possibility that accurate statements of resource footprint could be included in a program text to be checked by a compiler entirely locally, with the combination of separately-checked program parts requiring only that their resource claims are compatible. It also permits a simple treatment of shared-variable concurrency (dealt with in this paper) and, we anticipate, parameter aliasing (the subject of a future paper). Examples of the use of a similar logic to reason about shared-variable concurrency are given in Bornat et al. [4]; in particular there is a treatment of two versions of the readers and writers problem [8, 1], neither of which obeys the restrictions imposed by the side conditions of, for example, [16].

Despite the absence of side conditions, our logic can prove all the programs that conventional Hoare logic can prove. We have not abandoned the variable-assignment axiom and made assignment alter locations rather than program/logical variables, in the same way as separation logic [17, 13, 18] treats the heap. This has enabled our assertions to include the same sort of statements about the values of variables as are made in conventional Hoare logic, which would not be possible if 'stack' variables were forced into the heap.

Our work, however, draws on ideas from separation logic: program variables are treated as resource and resource claims are separated with $\star$. For clarity we exclude a treatment of the heap, though our logic can be extended to to deal with it: that is, we deal only with separation logic's 'stack'.

2. Variables as resource

Before setting out the formal details, we describe some intuitive notions behind variables as resource. Hoare logic does not allow us to describe the ownership of variables, as illustrated by the triple

$$\{y = 0\} \, x := 7 \, \{y = 0\}$$

which alters x but disingenuously avoids mentioning the fact. But x should be mentioned, because to execute $x := 7$ there must be a variable x in the stack. Furthermore, the assignment must *own* that variable, in the sense that no concurrent program can safely be permitted to read it. We must also know that variable y cannot be altered by some other program, else the assertion $y = 0$ cannot be assumed to be invariant. And then there is the matter of variable aliasing: x and y, distinct as names and as logical variables, must name distinct program variables – that is, distinct locations in the stack.

3. A logic of variables as resource

The syntax of assertions Φ is

$$\Phi \quad ::= \quad E = E \mid \mathbf{emp_s} \mid \mathrm{Own}_p(x) \mid \pi = \pi \mid$$
$$\Phi \Rightarrow \Phi \mid \text{false} \mid \forall X \cdot \Phi \mid \Phi \star \Phi \mid \Phi \rightarrowtail \Phi$$

We distinguish integer logical variables $X, Y, \ldots$, permission logical variables $p, \ldots$ and integer program variables $x, y, \ldots$. We do not quantify over the values of program variables. E and π range over integer and permission expressions respectively.

3.1. Model

Permissions [3] are fundamental to our logic. They allow us to describe the division of variables between the threads of a concurrent program, including read/write private ownership of variables (with total permission), read-only sharing of variables (with partial permissions in all the accessing threads) and correct access to critical variables (for example with ownership ascribed to the resource accessed in a conditional critical region). We can use them in a version of separation logic's frame rule to constrain access to global variables. They enable us to describe the variable usage of procedures. The side-conditions of Hoare logic are replaced by a careful description of the variable permissions required by each part of a program. Those descriptions allow more precise control than the old side conditions.

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:

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

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:

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

Separation logic divides the store into stack – the variables used by a program – and heap – dynamically allocated records – but does not give any formal treatment of

---

[1] Cancellative: $\iota \circledast \iota' = \iota \circledast \iota'' \Rightarrow \iota' = \iota''$.

the stack. In this paper we concentrate on program variables in the stack and logical and permission variables in the 'interpretation'.

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.

$$s : \mathcal{S} \stackrel{\text{def}}{=} \text{PVarNames} \rightharpoonup_{\text{fin}} \text{Int} \times \text{Perms}$$

$$i : \mathcal{R} \stackrel{\text{def}}{=} \text{LVarNames} \rightharpoonup_{\text{fin}} \text{Int} \cup \text{Perms}$$

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

$$[\![E1 + E2]\!]_{(s,i)} = [\![E1]\!]_{(s,i)} + [\![E2]\!]_{(s,i)}$$

$$[\![0]\!]_{(s,i)} = 0$$

$$[\![x]\!]_{(s,i)} = \begin{cases} s(x) & x \in dom(s) \\ \text{undefined} & \text{otherwise} \end{cases}$$

$$[\![X]\!]_{(s,i)} = i(X)$$

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

$$[\![\pi1 \circledast \pi2]\!]_{(s,i)} = [\![\pi1]\!]_{(s,i)} \circledast [\![\pi2]\!]_{(s,i)}$$

$$[\![\iota]\!]_{(s,i)} = \iota$$

$$[\![p]\!]_{(s,i)} = i(p)$$

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.

$\text{Own}_\pi(x)$ 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). $\text{Own}_\top(x)$ asserts total permission, i.e. ownership, and $\text{Own}_\_(x)$ means $\exists p \cdot (\text{Own}_p(x))$. $\mathbf{emp_s}$ asserts the empty stack, and true holds of any stack at all. Following separation logic, $(\star)$ combines stack assertions: $\text{Own}_\_(x) \star \text{Own}_\_(y)$ is a two-variable stack; $\text{Own}_\pi(x) \star \text{Own}_{\pi'}(x)$ is equivalent to $\text{Own}_{\pi \circledast \pi'}(x)$ and therefore $\text{Own}_\top(x) \star \text{Own}_\pi(x)$ is false; $\text{Own}_\_(x) \star \text{true}$ 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 $\text{Own}_\_(x) \star \text{true}$. 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.
$$x1_{\pi1}, \ldots, xn_{\pi n} \Vdash P \stackrel{\text{def}}{=}$$
$$(\text{Own}_{\pi1}(x1) \star \ldots \star \text{Own}_{\pi n}(xn)) \wedge P$$

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.

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 $x1_{\pi1}, \ldots, xn_{\pi n}$. 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 $Y = 3 \wedge \text{Own}_\top(x)$, then the precondition $Y = 3 \wedge \text{Own}_\top(x) \wedge Y = E$ 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.

---

[2] We subscript turnstiles to distinguish logics: $\vdash_{vr}$ for proof in the variables-as-resource logic in table 2; $\vdash_H$ for proof in Hoare logic in table 5.

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

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

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

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

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

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.

$$
\begin{aligned}
(z \leftrightarrow x)s \xrightarrow{C[z/x]}{}^{n}_{\rho} (z \leftrightarrow x)s' &\Rightarrow s \xrightarrow{C}{}^{n}_{\rho} s' \\
(z \leftrightarrow x)s \xrightarrow{C[z/x]}{}^{n}_{\rho} \text{unsafe} &\iff s \xrightarrow{C}{}^{n}_{\rho} \text{unsafe}
\end{aligned},
$$
(z fresh for C and ρ, $x \notin \text{dom}(s)$).

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

4

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

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

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

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

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

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

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

$$\frac{\Gamma, (\{\Psi1\}\, f(x; Y)\, \{\Psi2\})[z/x] \vdash_{vr} \{\Phi\}\ C\ \{\Phi'\}}{\Gamma, \{\Psi1\}\, f(x; Y)\, \{\Psi2\} \vdash_{vr} \{\Phi\}\ C\ \{\Phi'\}}\ (z\ \text{fresh for}\ \{\Psi1\}\, f(x; Y)\, \{\Psi2\})$$

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

Definition 9.
$\rho \vDash_n \Gamma \overset{\mathit{def}}{=}$ *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)**.**
$\Gamma \vDash_n \{\Phi\}C\{\Phi'\} \overset{\mathit{def}}{=}$
$\quad \forall \rho \cdot \big((\rho \vDash_n \Gamma) \Rightarrow (\rho \vDash_{n+1} \{\Phi\}C\{\Phi'\}) \big)$

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. $\qquad \square$

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:

$$X = E \wedge ((X = X \wedge \mathbf{emp_s}) \star E = E)$$
$$\not\Rightarrow\ (E = E \wedge \mathbf{emp_s}) \star E = E)$$

– 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 $\overset{\mathit{def}}{=}$
$\quad \forall s, s', i \cdot$
$\quad \big(((s, i) \vDash \Phi) \wedge \lfloor s \rfloor \subseteq \lfloor s' \rfloor \Rightarrow ((s', i) \vDash \Phi) \big)$
where $\lfloor s \rfloor = \{\langle x, v \rangle \mid \langle x, (v, p) \rangle \in s\}$

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

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

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

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

Note: $E \neq E' \iff \overset{s}{\neg}(E = E')$ is a tautology.
If we restrict the syntax of formulae our logic can use substitution of equals for equals.

5

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

$$s \xrightarrow{\text{skip}}{}^n_\rho s$$

$$\frac{s(x) = (\_, \top)}{s \xrightarrow{x := E}{}^n_\rho s \oplus \langle x, (\llbracket E \rrbracket_s, \top) \rangle}$$

$$\frac{\llbracket B \rrbracket_s = \text{true} \quad s \xrightarrow{C_{\text{true}}}{}^n_\rho s'}{s \xrightarrow{\text{if } B \text{ then } C_{\text{true}} \text{ else } C_{\text{false}} \text{ fi fi}}{}^n_\rho s'}$$

$$\frac{\llbracket B \rrbracket_s = \text{false} \quad s \xrightarrow{C_{\text{false}}}{}^n_\rho s'}{s \xrightarrow{\text{if } B \text{ then } C_{\text{true}} \text{ else } C_{\text{false}} \text{ fi fi}}{}^n_\rho s'}$$

$$\frac{s \xrightarrow{\text{if } B \text{ then } (C; \text{while } B \text{ do } C \text{ od}) \text{ else skip fi}}{}^n_\rho s'}{s \xrightarrow{\text{while } B \text{ do } C \text{ od}}{}^n_\rho s'}$$

$$\frac{s \xrightarrow{C1}{}^n_\rho s' \quad s' \xrightarrow{C2}{}^n_\rho s''}{s \xrightarrow{C1;C2}{}^n_\rho s''}$$

$$\frac{s \uplus \langle z, (\_, \top) \rangle \xrightarrow{C[z/x]}{}^n_\rho s' \uplus \langle z, (\_, \top) \rangle}{s \xrightarrow{\text{local } x \text{ in } C \text{ end}}{}^n_\rho s'} \text{ (fresh } z)$$

$$\frac{s \xrightarrow{C'}{}^n_{\rho \oplus \langle f, (y,z,C) \rangle} s'}{s \xrightarrow{\text{let } f(y;z)=C \text{ in } C' \text{ end}}{}^n_\rho s'}$$

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

$$\frac{\langle x, (\_, \top) \rangle \notin s}{s \xrightarrow{x := E}{}^n_\rho \text{ unsafe}} \quad \frac{\llbracket E \rrbracket_s \text{ is undefined}}{s \xrightarrow{x := E}{}^n_\rho \text{ unsafe}} \quad \frac{\llbracket B \rrbracket_s \text{ is undefined}}{s \xrightarrow{\text{if } B \text{ then } C1 \text{ else } C2 \text{ fi}}{}^n_\rho \text{ unsafe}}$$

Definition 16 (restricted formulae)**.**
$$\phi ::= E = E \mid \phi \land \phi \mid \phi \lor \phi \mid \phi \xRightarrow{s} \phi \mid \phi \rightarrowtail \phi \mid \pi = \pi \mid$$
$$\phi \star \phi \mid \forall X.\phi \mid \exists X.\phi \mid \text{false} \mid \text{true} \mid \overset{s}{\lnot} \phi$$

Lemma 17. *Restricted formulae are stack imprecise.*

Proof. Structural induction on ϕ. $\qquad\square$

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

Proof. By induction on structure of E' $\qquad\square$

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

Proof. By structural induction on ϕ. The $(\star)$ and $(\rightarrowtail)$ cases require lemma 14, and the $(=)$ case requires lemma 18. $\square$

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

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

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

Lemma 23. $\vDash (\overline{y_\pi} \Vdash \phi \star \psi) \iff \exists \overline{p1}, \overline{p2} \cdot ((\overline{y_{p1}} \Vdash \phi) \star (\overline{y_{p1}} \Vdash \psi)) \land (\overline{\pi} = \overline{p1 \circledast p2})$

Theorem 24 (Assignment by substitution)**.** *The assignment axiom in (2) is derivable.*

Proof.

$$\frac{\dfrac{\left\{ x_\top, \overline{y_{p1}} \Vdash E = X \right\} x := E \left\{ x_\top, \overline{y_{p1}} \Vdash x = X \right\}}{\left\{ (x_\top, \overline{y_{p1}} \Vdash E = X) \star \begin{pmatrix} (\overline{y_{p2}} \Vdash \phi[X/x]) \\ \land (\overline{p1} \circledast \overline{p1} = \overline{\pi}) \end{pmatrix} \right\} x := E}{\dfrac{\left\{ (x_\top, \overline{y_{p1}} \Vdash x = X) \star \begin{pmatrix} (\overline{y_{p2}} \Vdash \phi[X/x]) \\ \land (\overline{p1} \circledast \overline{p1} = \overline{\pi}) \end{pmatrix} \right\}}{\dfrac{\left\{ \exists \overline{p1}, \overline{p2}. (x_\top, \overline{y_{p1}} \Vdash E = X) \star \begin{pmatrix} (\overline{y_{p2}} \Vdash \phi[X/x]) \\ \land (\overline{p1} \circledast \overline{p1} = \overline{\pi}) \end{pmatrix} \right\} x := E}{\dfrac{\left\{ \exists \overline{p1}, \overline{p2}. (x_\top, \overline{y_{p1}} \Vdash x = X) \star \begin{pmatrix} (\overline{y_{p2}} \Vdash \phi[X/x]) \\ \land (\overline{p1} \circledast \overline{p1} = \overline{\pi}) \end{pmatrix} \right\}}{\dfrac{\{x_\top, \overline{y_\pi} \Vdash E = X \land \phi[E/x]\} x := E \{x_\top, \overline{y_\pi} \Vdash \phi\}}{\dfrac{\{\exists X \cdot x_\top, \overline{y_\pi} \Vdash E = X \land \phi[E/x]\} x := E \{\exists X \cdot x_\top, \overline{y_\pi} \Vdash \phi\}}{\{x_\top, \overline{y_\pi} \Vdash E = E \land \phi[E/x]\} x := E \{x_\top, \overline{y_\pi} \Vdash \phi\}}}}}}}$$

The first use of the rule of consequence requires

$$x_\top, \overline{y_{\overline{\pi}}} \Vdash E = X \wedge \phi[E/x]$$
$$\Rightarrow x_\top, \overline{y_{\overline{\pi}}} \Vdash E = X \wedge \phi[X/x][E/X] \qquad (X \notin \phi)$$
$$\Rightarrow x_\top, \overline{y_{\overline{\pi}}} \Vdash (E = X) \star \phi[X/x] \qquad \text{(Lemmas 19,14)}$$
$$\Rightarrow \exists p, p', \overline{p1}, \overline{p2}.$$
$$\qquad ((x_p, \overline{y_{\overline{p1}}} \Vdash E = X) \star (x_{p'}, \overline{y_{\overline{p2}}} \Vdash \phi[X/x]))$$
$$\qquad \wedge (p \circledast p' = \top) \wedge (\overline{p1} \circledast \overline{p2} = \overline{\pi}) \qquad \text{(Lemma 23)}$$
$$\Rightarrow \exists \overline{p1}, \overline{p2}. ((x_\top, \overline{y_{\overline{p1}}} \Vdash E = X) \star (\overline{y_{\overline{p2}}} \Vdash \phi[X/x]))$$
$$\qquad \wedge (\overline{p1} \circledast \overline{p2} = \overline{\pi}) \qquad \text{(Lemma 21)}$$
$$\Rightarrow \exists \overline{p1}, \overline{p2}. (x_\top, \overline{y_{\overline{p1}}} \Vdash E = X)$$
$$\qquad \star ((\overline{y_{\overline{p2}}} \Vdash \phi[X/x]) \wedge (\overline{p1} \circledast \overline{p2} = \overline{\pi}))$$

and

$$\exists \overline{p1}, \overline{p2}. (x_\top, \overline{y_{\overline{p1}}} \Vdash x = X)$$
$$\qquad \star ((\overline{y_{\overline{p2}}} \Vdash \phi[X/x]) \wedge (\overline{p1} \circledast \overline{p2} = \overline{\pi}))$$
$$\Rightarrow x_\top, \overline{y_{\overline{\pi}}} \Vdash (x = X) \star \phi[X/x] \quad \text{(Lemma 21)}$$
$$\Rightarrow x_\top, \overline{y_{\overline{\pi}}} \Vdash x = X \wedge \phi[X/x] \quad \text{(Lemma 14)}$$
$$\Rightarrow x_\top, \overline{y_{\overline{\pi}}} \Vdash \phi \qquad \text{(Lemma 19)}$$

The second use of the rule of consequence requires

$$E = E \Rightarrow \exists X \cdot E = X$$

$\square$

5. Encoding Hoare logics

We must at least be able to prove all assertions provable in Hoare logic. We demonstrate this by showing that Hoare logic proofs can be translated into our logic. We present a translation of a Hoare logic with reference and value parameters in procedure definitions. We use ϕ and ψ to range over Hoare logic assertions since there is an implicit translation to restricted formulae: $\overset{s}{\Rightarrow}$ for $\Rightarrow$, $\overset{s}{\neg}$ for $\neg$.

A Hoare-logic function context $\mathbb{F}$ (cf. Γ) is a set of specifications $\{\phi\}\, f(x; y)[\overline{u}; \overline{v}]\, \{\psi\}$ where

- f is a function name, x a reference-parameter name and y a value-parameter name;
- ϕ is the precondition and ψ the postcondition of $f(x; y)$;
- $\overline{u}$ is a set of the names of the global variables modifed by $f(x; y)$ and $\overline{v}$ a set of the names of global variables it reads;
- $\overline{u} \subseteq \overline{v}$.

Table 4 defines $\mathrm{mods}(\mathbb{F}, C)$, the variables written by C, and $\mathrm{free}(\mathbb{F}, C)$, its free variables: because of the complexities of the let definition we require two definitions for function call but, because let declares functions one at a time, we do not need a fixed-point iteration. Table 5 gives the rules of the Hoare logic which we encode.

Lemma 25 (The logics are equivalent on defined assertions)**.**

$$\mathrm{FV}(\phi) \subseteq \mathrm{dom}(s) \Rightarrow$$
$$\left(\, (\lfloor s \rfloor, i) \models_H \phi \iff (s, i) \models \phi \, \right)$$

where $\models$ is the forcing semantics given in table 1, and $\models_H$ is the forcing semantics of Hoare logic.

Proof. Structural induction on ϕ. The interesting case is $\star$, which requires that ϕ is stack imprecise. $\square$

Definition 26 (Supporting write and read variables)**.**

$$\mathrm{supports}_{\overline{p}}(\overline{u}; \overline{v}) \overset{def}{=}$$
$$(\overline{u}_1)_\top, \ldots, (\overline{u}_m)_\top, (\overline{w}_1)_{\overline{p}_1}, \ldots, (\overline{w}_n)_{\overline{p}_n}$$
$$\text{where } \overline{w} = \overline{v} \setminus \overline{u}$$

Definition 27 (Supporting a command)**.**

$$\mathrm{supports}_{\overline{p}}(\mathbb{F}, C) \overset{def}{=} \mathrm{supports}_{\overline{p}}(\mathrm{mods}(\mathbb{F}, C); \mathrm{free}(\mathbb{F}, C))$$

In Hoare logic program variables and logical variables are conflated. In our translation of $\{\phi\}\, C\, \{\psi\}$ we turn all the free variables of ϕ and ψ that are not used in C into logical variables.

Definition 28 (Triple translation)**.**

$$[\![\{\phi\}\, C\, \{\psi\}]\!]_\mathbb{F} \overset{def}{=} \{O \Vdash \phi[\overline{U}/\overline{u}]\}\ C\ \{O \Vdash \psi[\overline{U}/\overline{u}]\}$$
$$\text{where}\quad O = \mathrm{supports}_{\overline{p}}(\mathbb{F}, C);$$
$$\overline{u} = \mathrm{FV}(\phi, \psi) \setminus \mathrm{vars}(O);$$
$$\text{fresh } \overline{p}, \overline{U}$$

(Here $\overline{p}$ and $\overline{U}$ are sets of fresh logical variables, implicitly quantified at the level of the triple: that is, because of the semantics of triples, the fresh $\overline{p}, \overline{U}$ can be thought of as universally quantified.

$$\forall \overline{p}, \overline{U} \cdot \{O \Vdash \phi[\overline{U}/\overline{u}]\}\ C\ \{O \Vdash \psi[\overline{U}/\overline{u}]\}$$

Clearly, the translation is deterministic.)

Although our translation replaces some integer program variables with new logical variables, we can always retrieve the original specification by extending O and using the frame rule to enforce an invariant which equates the values of new and old variables.

Lemma 29 (From proof with logical variables infer proof with program variables.)**.**

7

Table 4. Modified and free variables of commands $\quad mods(\mathbb{F}, C)$**,** $free(\mathbb{F}, C)$

C	$mods(\mathbb{F}, C)$	$free(\mathbb{F}, C)$
$x := E$	$\{x\}$	$FV(E) \cup \{x\}$
$C_1; C_2$	$mods(\mathbb{F}, C_1) \cup mods(\mathbb{F}, C_2)$	$free(\mathbb{F}, C_1) \cup free(\mathbb{F}, C_2)$
skip	$\varnothing$	$\varnothing$
while B do C od	$mods(\mathbb{F}, C)$	$FV(B) \cup free(\mathbb{F}, C)$
local x in C end	$mods(\mathbb{F}, C) \setminus \{x\}$	$free(\mathbb{F}, C) \setminus \{x\}$
if B then C_1 else C_2 fi	$mods(\mathbb{F}, C_1) \cup mods(\mathbb{F}, C_2)$	$FV(B) \cup free(\mathbb{F}, C_1) \cup free(\mathbb{F}, C_2)$
let $f(x;y) = C$ in C' end where $\mathbb{F}' = \mathbb{F}, \{\_\}f(x;y)[\overline{u}; \overline{v}]\{\_\};$ $\overline{u} = mods(\mathbb{F}, C) \setminus \{x, y\};$ $\overline{v} = free(\mathbb{F}, C) \setminus \{x, y\}$	$mods(\mathbb{F}', C')$	$free(\mathbb{F}', C')$
$f(x; E)$ (normal case, $\{\_\}f(x;y)[\overline{u}; \overline{v}]\{\_\} \in \mathbb{F}$)	$\{x\} \cup \overline{u}$	$\{x\} \cup FV(E) \cup \overline{v}$
$f(x; E)$ (bootstrap case, $\{\_\}f(\_; \_)[\_; \_]\{\_\} \notin \mathbb{F}$)	$\{x\}$	$\{x\} \cup FV(E)$

Table 5. Hoare logic rules: $\quad \mathbb{F} \vdash_H \{\phi\}\ C\ \{\phi\}$

$$\overline{\mathbb{F} \vdash_H \{\phi[E/x]\}\ x := E\ \{\phi\}} \qquad \overline{\mathbb{F} \vdash_H \{\phi\}\ \text{skip}\ \{\phi\}}$$

$$\frac{\mathbb{F} \vdash_H \{\phi \wedge B\}\ C_1\ \{\psi\} \quad \mathbb{F} \vdash_H \{\phi \wedge \neg B\}\ C_2\ \{\psi\}}{\mathbb{F} \vdash_H \{\phi\}\ \text{if } B \text{ then } C_1 \text{ else } C_2 \text{ fi}\ \{\psi\}} \qquad \frac{\mathbb{F} \vdash_H \{\phi \wedge B\}\ C\ \{\phi\}}{\mathbb{F} \vdash_H \{\phi\}\ \text{while } B \text{ do } C \text{ od}\ \{\phi \wedge \neg B\}}$$

$$\frac{\phi' \Rightarrow \phi \quad \mathbb{F} \vdash_H \{\phi\}\ C\ \{\psi\} \quad \psi \Rightarrow \psi'}{\mathbb{F} \vdash_H \{\phi'\}\ C\ \{\psi'\}} \qquad \frac{\mathbb{F} \vdash_H \{\phi\}\ C\ \{\phi'\}}{\mathbb{F} \vdash_H \{\phi \wedge \psi\}\ C\ \{\phi' \wedge \psi\}}\ mods(\mathbb{F}, C) \cap FV(\psi) = \varnothing$$

$$\frac{\mathbb{F} \vdash_H \{\phi\}\ C\ \{\psi\}}{\mathbb{F} \vdash_H \{\exists x.\phi\}\ C\ \{\exists x.\psi\}}\ x \notin free(\mathbb{F}, C) \qquad \frac{\mathbb{F} \vdash_H \{\phi\}\ C[y/x]\ \{\psi\}}{\mathbb{F} \vdash_H \{\phi\}\ \text{local } x \text{ in } C \text{ end}\ \{\psi\}}\ y \text{ fresh}$$

$$\overline{\mathbb{F} \vdash_H \{\phi[w, Y/x, y] \wedge E = Y\}\ f(w; E)\ \{\psi[w, Y/x, y]\}}\ w \notin \overline{u}, \overline{v}; \{\phi\}f(x;y)[\overline{u}; \overline{v}]\{\psi\} \in \mathbb{F}$$

$$\frac{\mathbb{F}' \vdash_H \{\phi\}\ C\ \{\psi\} \quad \mathbb{F}' \vdash_H (\{\phi'\}\ \text{local } z \text{ in } z := y; C'[z/y] \text{ end}\ \{\psi'\})[w/x]}{\mathbb{F} \vdash_H \{\phi\}\ \text{let } f(x;y) = C' \text{ in } C \text{ end}\ \{\psi\}}\ \begin{array}{l}\mathbb{F}' = \mathbb{F}, \{\phi'\}f(x;y)[\overline{u}; \overline{v}]\{\psi'\};\\ \overline{u} = free(\mathbb{F}, C') \setminus \{x, y\};\ \overline{v} = mods(\mathbb{F}, C') \setminus \{x, y\}\\ \text{fresh } w\end{array}$$

Proof.

$$\frac{\dfrac{\{O \Vdash \phi[Z/z]\}\ C\ \{O \Vdash \psi[Z/z]\}}{\left\{\begin{array}{l}(z_\pi \Vdash Z = z)\\ \star (O \Vdash \phi[Z/z])\end{array}\right\}\ C\ \left\{\begin{array}{l}(z_\pi \Vdash Z = z)\\ \star (O \Vdash \psi[Z/z])\end{array}\right\}}}{\dfrac{\left\{\exists Z \cdot \begin{pmatrix}(z_\pi \Vdash Z = z)\\ \star (O \Vdash \phi[Z/z])\end{pmatrix}\right\}\ C\ \left\{\exists Z \cdot \begin{pmatrix}(z_\pi \Vdash Z = z)\\ \star (O \Vdash \psi[Z/z])\end{pmatrix}\right\}}{\{O, z_\pi \Vdash \phi\}\ C\ \{O, z_\pi \Vdash \psi\}}}$$

$\square$

Definition 30 (Translation of procedure environment)**.**

$$[\![\mathbb{F}]\!] \stackrel{def}{=} \left\{ \left[\!\!\left[\begin{array}{c} \{\phi[Y/y]\}\\ f(x;Y)\\ \{\psi[Y/y]\} \end{array} \right]\!\!\right]_{\mathbb{F}} \ \middle|\ \{\phi\}f(x;y)[\overline{u}; \overline{v}]\{\phi\} \in \mathbb{F} \right\}$$

Note: For convenience, we assume that triple translation treats Y in $f(x; Y)$ as a constant.

Theorem 31 (Completeness of encoding)**.**

$$(\mathbb{F} \vdash_H \{\phi\}\ C\ \{\psi\}) \Rightarrow ([\![\mathbb{F}]\!] \vdash_{vr} [\![\{\phi\}\ C\ \{\psi\}]\!]_{\mathbb{F}})$$

Proof. By induction on the Hoare-logic derivation. $\square$

Theorem 32 (Soundness of encoding)**.**

$$([\![\mathbb{F}]\!] \vDash_{vr} [\![\{\phi\}\ C\ \{\psi\}]\!]_{\mathbb{F}}) \Rightarrow (\mathbb{F} \vDash_H \{\phi\}\ C\ \{\psi\})$$

Proof.

$$(\mathbb{F} \vDash_{vr} [\![\{\phi\}\ C\ \{\psi\}]\!]_{\mathbb{F}})$$
$$\Rightarrow (\mathbb{F} \vDash_{vr} \{O, O' \Vdash \phi\}\ C\ \{O, O' \Vdash \psi\})$$

8

where $O = \mathrm{supports}_{\overline{p}}(\mathbb{F}, C)$, and $\mathrm{vars}(O, O') \supseteq \mathrm{FV}(\phi, \psi)$ follows directly from repeated application of lemma 29. Then

$$(\mathbb{F} \vDash_{vr} \{O, O' \Vdash \phi\}\ C\ \{O, O' \Vdash \psi\})$$
$$\Rightarrow (\mathbb{F} \vDash_{H} \{\phi\}\ C\ \{\psi\})$$

follows directly from lemma 25. $\qquad\square$

6. Concurrency

Hoare-logic concurrency rules have complex side conditions and restrictions constraining the use of variables. In our logic we do not need any of that. The rules are given with respect to a resource context, Δ, which maps a resource identifier b to its corresponding invariant. The invariants must be precise [6]. Table 6 gives the rules.

6.1. Soundness

Brookes has shown this logic to be sound [5].

6.2. Translation

We can translate Brookes's rules for concurrent separation logic [6] into our own (we omit his treatment of the heap). The key to his soundness proof is the notion of critical variable. A variable is critical if it is modified in one thread and free in another: in $x := y \| y := z$, for example, y is critical.

Each critical variable is associated with a *resource* and each access must be within a critical region for that resource. A resource context $\mathbb{R}$ maps a resource name b to a critical-variable list $\overline{u}$ and an invariant ψ. There are two operations on these contexts: (1) $\mathrm{crit}(\mathbb{R})$ delivers the critical variables in $\mathbb{R}$; and (2) $\mathrm{FV}(\mathbb{R})$ the variables free in the invariants as well as all critical variables. Brookes's parallel rule has a side condition:

$$\mathrm{FV}(\Phi 1, \Phi 1') \cap \mathrm{mods}(C2) = \varnothing$$
$$\mathrm{FV}(\Phi 2, \Phi 2') \cap \mathrm{mods}(C1) = \varnothing$$
$$\mathrm{FV}(C1) \cap \mathrm{mods}(C2) \subseteq \mathrm{crit}(\mathbb{R})$$
$$\mathrm{FV}(C2) \cap \mathrm{mods}(C1) \subseteq \mathrm{crit}(\mathbb{R})$$

The $x := E$ rule has the side condition $x \notin \mathrm{FV}(\mathbb{R})$. There is also an additional constraint on the well-formedness of judgements $\mathbb{R} \vdash_{H} \{\Phi\}\ C\ \{\Phi'\}$: the critical variables may not be mentioned in the pre- or post-condition, i.e. $crit(\mathbb{R}) \cap \mathrm{FV}(\Phi, \Phi') = \varnothing$.

Definition 33.
$$\mathrm{supports}_{\overline{p}}(\mathbb{R}, C) \stackrel{def}{=}$$
$$\mathrm{supports}_{\overline{p}}((\mathrm{mods}(C) \setminus \mathrm{FV}(\mathbb{R})); (\mathrm{free}(C) \setminus \mathrm{crit}(\mathbb{R})))$$

Definition 34 (Triple translation).
$$[\![\{\phi\}\ C\ \{\psi\}]\!]_{\mathbb{R}} \stackrel{def}{=} \{O \Vdash \phi[\overline{U}/\overline{u}]\}\ C\ \{O \Vdash \psi[\overline{U}/\overline{u}]\}$$
where $O = \mathrm{supports}_{\overline{p}}(\mathbb{R}, C)$, $\overline{u} = \mathrm{FV}(\phi, \psi) \setminus \mathrm{vars}(O)$, *and* $\overline{p}$, $\overline{U}$ *are sets of fresh logical variables.*

Definition 35 (Context translation, $[\![\mathbb{R}]\!]$). *We translate each element of the context to*

$$[\![r[\overline{u}] : \psi]\!] = r : \mathrm{supports}_{\overline{p}}(\overline{u}, \mathrm{FV}(\psi)) \Vdash \psi$$

Lemma 36.

$$\mathrm{supports}_{\overline{p}}(\mathbb{R}, C) =$$
$$\mathrm{supports}_{\overline{p}}(\overline{u}, \mathrm{FV}(\psi)) \star \mathrm{supports}_{\overline{p}}(\mathbb{R} \uplus \langle b, (\overline{u}, \psi) \rangle, C)$$

Theorem 37 (Completeness of encoding).

$$\mathbb{R} \vdash_{H} \{\phi\}\ C\ \{\psi\} \quad \Rightarrow \quad [\![\mathbb{R}]\!] \vdash_{vr} [\![\{\phi\}\ C\ \{\psi\}]\!]_{\mathbb{R}}$$

Proof. By induction on the derivation for Brookes's rules. $\qquad\square$

Theorem 38 (Soundness of encoding).

Proof. Same as theorem 32 $\qquad\square$

7. Conclusions

We have a logic which admits translations of all Hoare logic proofs and in which there are no side conditions on the use of variables or restrictions on the action of concurrent programs. In addition we are able (though not within the space constraints of this presentation) to deal with the heap in the same way. By working through several examples, Bornat et al. have previously shown that this kind of logic deals conveniently with the verification of shared-variable concurrency programs [4]. Their logic can be translated into our own, and its soundness is a consequence of the soundness of our own.

In all other previous Hoare logics a simultaneous treatment of concurrency, procedure call and the heap requires complex side conditions on the use of variables, as well as restrictions on the action of programs which are extremely difficult to check in a mechanical proof tool. Smallfoot [2], for example, uses a treatment of concurrency based on Brookes' and O'Hearn's treatment in separation logic, and must make a completely global static analysis when dealing with the restrictions on concurrent programs. The analysis takes several pages to describe, and is extremely intricate to implement. No such analysis would be required in a tool based on our new logic.

Table 6. Variables-as-resource rules for concurrency

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

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

References

[1] G. Andrews. *Concurrent Programming: Principles and Practice*. Benjamin Cummings, 1991.

[2] J. Berdine, C. Calcagno, and P. W. O'Hearn. Modular automatic assertion checking with separation logic. Draft, Nov. 2005.

[3] R. Bornat, C. Calcagno, P. O'Hearn, and M. Parkinson. Permission accounting in separation logic. In *POPL '05: Proceedings of the 32nd ACM SIGPLAN-SIGACT sysposium on Principles of programming languages*, pages 259–270, New York, NY, USA, Jan. 2005. ACM Press.

[4] R. Bornat, C. Calcagno, and H. Yang. Variables as resource in separation logic. Presented at MFPS XXI, Birmingham, May 2005. To appear in Electronic Notes in Computer Science, 2005.

[5] S. Brookes. Variables as resource for shared-memory programs: Semantics and soundness. In *Proceedings of MFPS XXII*. Elsevier ENTCS., May 2006.

[6] S. D. Brookes. A semantics for concurrent separation logic. In *CONCUR'04: 15th International Conference on Concurrency Theory*, volume 3170 of *Lecture Notes in Computer Science*, pages 16–34, London, Aug. 2004. Springer. Extended version to appear in *Theoretical Computer Science*.

[7] R. Cartwright and D. Oppen. Unrestricted procedure calls in hoare's logic. In *POPL '78: Proceedings of the 5th ACM SIGACT-SIGPLAN symposium on Principles of programming languages*, pages 131–140, New York, 1978. ACM Press.

[8] P. J. Courtois, F. Heymans, and D. L. Parnas. Concurrent control with "readers" and "writers". *Commun. ACM*, 14(10):667–668, 1971.

[9] D. Gries and G. Levin. Assignment and procedure call proof rules. *ACM Transactions on Programming Languages and Systems*, 2(4), Oct. 1980.

[10] C. A. R. Hoare. An axiomatic basis for computer programming. *Commun. ACM*, 12(10):576–580, 1969.

[11] C. A. R. Hoare. Towards a theory of parallel programming. *Operating Systems Techniques*, 1971.

[12] S. S. Ishtiaq and P. W. O'Hearn. BI as an assertion language for mutable data structures. In *Symposium on Principles of Programming Languages*, pages 14–26, 2001.

[13] P. O'Hearn, J. Reynolds, and H. Yang. Local reasoning about programs that alter data structures. In L. Fribourg, editor, *CSL 2001*, pages 1–19. Springer-Verlag, 2001. LNCS 2142.

[14] P. W. O'Hearn. Resources, concurrency and local reasoning. To appear in *Theoretical Computer Science*; preliminary version published as [15].

[15] P. W. O'Hearn. Resources, concurrency and local reasoning. In *CONCUR'04: 15th International Conference on Concurrency Theory*, volume 3170 of *Lecture Notes in Computer Science*, pages 49–67, London, Aug. 2004. Springer. Extended version is [14].

[16] S. Owicki and D. Gries. An axiomatic proof technique for parallel programs. *Acta Informatica*, 19:319–340, 1976.

[17] J. C. Reynolds. Intuitionistic reasoning about shared mutable data structure. In J. Davies, B. Roscoe, and J. Woodcock, editors, *Millennial Perspectives in Computer Science*, pages 303–321. Palgrave, 2000.

[18] J. C. Reynolds. Separation logic: A logic for shared mutable data structures. In *LICS '02: Proceedings of the 17th Annual IEEE Symposium on Logic in Computer Science*, pages 55–74, Washington, DC, USA, 2002. IEEE Computer Society.

[19] C. Strachey. Fundamental concepts in programming languages. *Higher Order Symbolic Computation*, 13(1-2):11–49, 2000.