# Variables as Resource in Separation Logic

Richard Bornat

*School of Computing Science, Middlesex University, London, UK.*
*R.Bornat@mdx.ac.uk*

Cristiano Calcagno

*Department of Computing, Imperial College London, London, UK.*
*ccris@doc.ic.ac.uk*

Hongseok Yang

*ERC-ACI, Seoul National University, Seoul, Korea.* *hyang@ropas.snu.ac.kr*

**Abstract**

Separation logic [20,21,14] began life as an extended formalisation of Burstall's treatment of list-mutating programs [8]. It rapidly became clear that there was more that it could say: O'Hearn's discovery [13] of ownership transfer of buffers between threads and Boyland's suggestion [5] of permissions to deal with variable and heap sharing pointed the way to a treatment of safe resource management in concurrent programs. That treatment has so far been incomplete because it deals only with heap cells and not with with (stack) variables as resource.

Adding 'variable contexts' — in the simplest case, lists of owned variables — to assertions in Hoare logic allows a resource treatment of variables. It seems that a formal treatment of aliasing is possible too. It gives a complete formal treatment of critical sections (for the first time, so far as I am aware).

*Key words:* separation, variables, verification, proof

## 1 Background

Separation logic, pre permissions, is described in [21,14]. It began as an expansion of Burstall's treatment of lists [8]. Burstall recognised that if lists in the store are *separated*, we can reason about mutations to each list *separately*. Reynolds, in [20], extended that idea to mutations of all kinds of heap data structure. He, O'Hearn and others made the idea of separation central to a logic which is a model of BI [17,19] and an extension of Hoare logic. The current state of separation logic, and what you can do with it, is described in detail in [15] and [7]. I give a very brief summary here of relevant points.

Separation logic has the normal connectives and quantifiers of classical predicate calculus, plus

- **emp** is an empty heap;
- $E \mapsto E'$ is a heap of one cell with address $E$ and contents $E'$ ($E$ and $E'$ must be 'pure' expressions made up of constants and variable names, and thus cannot mention the heap);
- $E \mapsto \_$ is shorthand for $\exists k \cdot (E \mapsto k)$;
- $A \star B$ is a heap which can be separated into two parts, one described by $A$ and the other by $B$;
- $A \wedge B$ is a heap which is described by $A$ and at the same time by $B$;
- $A \mathrel{-\!\!\star} B$ is a heap which, if we added a separate portion satisfying $A$, would satisfy $B$.

Proof-theoretically ($\star$) is multiplicative conjunction (one part of the context proves $A$, the other $B$) whereas ($\wedge$) is additive conjunction (the whole context proves $A$ and the same context proves $B$). Similarly, ($\mathrel{-\!\!\star}$) is multiplicative implication and ($\rightarrow$) is additive implication. In practice so far most researchers don't use ($\mathrel{-\!\!\star}$) very much.

Building separation into the logic makes it possible to make very elegant proofs of heap-mutating programs, making earlier attempts (e.g. my own in [3]) look clumsy and over-complicated. But it was O'Hearn's invention of ownership transfer [15] that really gave the logic an edge: it is possible to show, using the idea of separation, how ownership of a heap buffer can be transferred from one concurrent thread to another. Brookes, in [7], showed that the idea is sound. The notion of permissions, in [4] — not yet proved sound, but with a convincing model — allows partial degrees of ownership and in particular read-only shared heap buffers.

Not everything is settled yet. In particular there are difficulties with variables — the 'stack', in Reynolds' terminology. The ($\star$) operator works on the heap, but the stack is dealt with by side conditions. The frame rule, for example, which allows us to focus on the resource $Q$ and $R$ that a command needs to do its work, is

$$\frac{\{Q\}\, C\, \{R\}}{\{P \star Q\}\, C\, \{P \star R\}} \ (\mathit{modifies}\ C\ \cap \mathit{vars}\ P = \emptyset) \tag{1}$$

The *frame property* of the logic is subtle: informally, if a command $C$ is safe (doesn't go wrong) in a given heap, then the result of executing it in a larger heap can be tracked to *some* execution in the original heap [22]. If $P$ is separate from $Q$, and $C$ transforms $Q$ into $R$ then, by the frame property, surely when $C$ finishes we have $R$ and — separately and therefore untouched — still $P$. Well, yes: but if $C$ alters one of the variables that appear in $P$ then the meaning of $P$ changes; the side condition outlaws just that possibility, and the rule is sound. That side condition is not easily checked: modifying assignments could appear in $C$, provided they aren't reachable starting from a state described by $Q$; if $C$ contains procedure calls then the modification constraint must be applied to those procedures.

Concurrency is dealt with by two associated rules and by the notion of named *resource bundles*. The concurrency rule:

$$\frac{\{Q_1\}\,C_1\,\{R_1\} \quad \cdots \quad \{Q_n\}\,C_n\,\{R_n\}}{\{Q_1 \star \cdots \star Q_n\}\,(C_1 \parallel \cdots \parallel C_n)\,\{R_1 \star \cdots \star R_n\}} \tag{2}$$

has two side conditions: a variable changed in one thread can't appear in another unless it is owned by a resource bundle; and thread $C_i$ must not modify variables free in $Q_j$ or $R_j$ when $j \neq i$.

The CCR rule: [1]

$$\frac{\{(Q \star I_b) \wedge G\}\,C\,\{R \star I_b\}}{\{Q\}\,\text{with } b \text{ when } G \text{ do } C \text{ od}\,\{R\}} \tag{3}$$

also has two side conditions. Variables owned by resource bundle $b$ can't appear in the program outside the guard $G$ and command $C$ of a CCR (that, at least, is a syntactic condition that a compiler could easily check); and no other process can modify variables free in $Q$ and $R$ (subtle, global, hard to check, and we shall see what it means in section 11).

CCRs aren't easy to implement, but they are easy to reason about, and O'Hearn points out in [15] that it's convenient to treat the easy-to-implement mutex as a special form of CCR:

$$\begin{array}{l} \mathrm{P}(m) \text{ is “with } m \text{ when } m = 1 \text{ do } m := 0 \text{ od”} \\ \mathrm{V}(m) \text{ is “with } m \text{ when true do } m := 1 \text{ od”} \end{array} \tag{4}$$

(we shall see a similar treatment of counting semaphores later, whose command-bodies are $n{-}{-}$ and $n{+}{+}$ instead of $m := 0$ and $m := 1$). This treatment inverts the usual treatment of semaphores as 'keep-out' barriers and makes them 'come-in' stores of resource. That's an important insight, and it is exploited in all the proofs below.

Although in this paper I'm trying to polish the frame and concurrency rules by eliminating side conditions, it's important to acknowledge what's already marvellous about them. The frame property lies behind the frame rule: if program components don't exceed their specified resource footprints then we can focus on the footprint and ignore everything else. The stage magic

[1] CCR for Conditional Critical Region: $C$ (command) is the critical region, $G$ (guard) is the condition. CCRs were introduced by Hoare in [12]. Hoare called $b$ a 'resource'; I call it a 'resource bundle', or 'bundle' for short. The treatment here, taken from O'Hearn [15], is different in other details from Hoare's but very close in spirit; it is significantly simpler than that of Owicki and Gries [18].

practised by new and dispose, with axioms

$$
\begin{array}{l}
\{\mathbf{emp}\}\, x := \mathrm{new}()\, \{x \mapsto \_\} \\
\{E \mapsto \_\}\ \mathrm{dispose}\, E\, \{\mathbf{emp}\}
\end{array}
\tag{5}
$$

preserves the frame property even when heap resource is created and reclaimed.

In the concurrency rule we can reason about threads separately if we have the frame property and if every thread plays by the resource-bundle-invariant rules. The *separation property* is that separated well-behaved threads stay separated. And that, crucially, allows an explanation of how critical sections work so far as the heap is concerned: if I have a heap resource $E \mapsto \_$ which I got from a resource bundle using a CCR, I can be sure that you can't have it at the same time. If the critical section needs that heap resource to do its work, it's *impossible* that we could both be resourced to execute it at the same time. So if we are separate well-behaved threads – 'well-behaved' means staying within our resource footprints – then resource exclusivity implies critical-section mutual exclusion. That argument is new, so far as I'm aware, and it seems to me to be the basis of the best available formal explanation of how critical sections work.

The argument works fine for heap buffers, but what do we do about variables? Mutual exclusion of critical sections often depends on ownership of variables (consider, for example, the variable $c$ in figure 4, which must only be used during the critical sections between $P(m)$ and $V(m)$). Variables ought to be resource, treated formally by the logic and not mumbled over in side conditions.

There seem to be two approaches to the problem. One is to "put the stack in the heap" — i.e. treat variables in the same way that we treat heap buffers. That approach, though it would have considerable advantages if it were successful, hasn't yet led to an elegant logic. In particular it seems that we would have to give up the beautiful simplicity of Hoare's variable-assignment axiom (7), and our assertions would become much more complicated.

The other approach is to run at it bull-headed, as they say in Lancashire, and describe variable ownership in such a way that it isn't affected by substitution in the variable-assignment rule. That is the approach taken here.

## 2 The problem summarised

Separation logic permits only a restricted range of assignments. If $E$ is a *pure expression*, mentioning only variables and constants but not the heap, we are allowed

- $x := E$ (variables to variable);
- $x := [E]$ (heap to variable);

- $[E] := E'$ (variables to heap);
- $x := \text{new}()$ (expand the heap and assign to a variable).

All other formulae in the language — in conditionals, loops and procedure arguments — must be pure expressions. The frame rule then lets us focus on the heap resource used by assignments, through the 'small axioms' (the axiom for new is given in (5)):

$$\begin{array}{l} \{E \mapsto E'\}\, x := [E]\, \{E \mapsto E' \wedge x = E'\} \\ \qquad\qquad\qquad\qquad \text{provided } x \text{ does not occur free in } E \text{ or } E' \\ \{E \mapsto \_\}\, [E] := E'\, \{E \mapsto E'\} \end{array} \tag{6}$$

The restriction to pure expressions $E$ and $E'$ means that we don't have to worry about aliasing in heap mutation, and ($\star$)-separation does the rest. There are 'backward' (weakest-precondition) versions of these axioms which are a little more complicated but don't have side conditions. The side condition isn't important for this discussion, though, and for simplicity of explanation I'm going to stick with the forward versions throughout the paper.

Pure-expression-to-variable assignment doesn't use the heap, so it can use the backward-reasoning Hoare axiom

$$\{R^x_E\}\, x := E\, \{R\}, \text{ provided } x \text{ is not aliased in } R. \tag{7}$$

This allows easy mechanical calculation, which is nice, gives a weakest precondition, which is even nicer, and has no side condition worth worrying about since separation logic has no means of aliasing stack variables. In particular, because heaps are partial maps from Nat to Int, it works well with $E \mapsto E'$ assertions in $R$. For example:

$$\{x + 1 \mapsto 2(x + 1)\}\, x := x + 1\, \{x \mapsto 2x\} \tag{8}$$

The assignment doesn't affect ownership of the heap — we own one cell before the assignment, and we own the same cell with the same contents afterwards — but it does affect the way we describe that ownership. Beforehand we describe the cell by giving the address $x + 1$; afterwards the same address is described by $x$. This is all very fine, completely obvious, and just as it should be. It supports the reading of $\mapsto$ as ownership [15], and it even allows it to be read as a permission [4].

## *2.1 Assignment doesn't affect owership of variables*

I want to extend the notions of ownership and permission to stack variables. It's important to recognise a principle: *assignment doesn't affect ownership.* To avoid races, if you execute $x := x + 1$ you must own the variable $x$. To reason about how that assignment interacts with the postcondition $x = 2y$

you must also have permission to read $y$ — once again, to avoid races, to stop some other thread/process writing $y$ while you're pausing for breath.

$$\{x + 1 = 2y\}\, x := x + 1\, \{x = 2y\} \tag{9}$$

Before and after the assignment you must own $x$ and be able to read $y$. You might think you could deduce that from the assignment and the free variables of the precondition and postcondition, but not so. If it were so, then assignment would affect ownership:

$$\{y = 3\}\, x := y\, \{x = 3\} \tag{10}$$

It seems to me that we can't hope to deduce ownership and permission constraints from the equalities and inequalities of ordinary Hoare logic assertions.

## 3 Groping for a solution

The first approach which comes to mind is to get rid of the stack, to put variables in the heap and to use ($\mapsto$) and ($\star$) to deal with them. That, after all, corresponds to the way the hardware does it (though on many machines the registers don't have addresses, and 'variables' in 'the stack' are more like machine registers than anything else, so perhaps it's not a complete answer). But to date there hasn't been an elegant solution based on that idea. In particular it becomes difficult to write simple assertions like $x > y$: instead you have to write stuff like $\exists X, Y \cdot ((x \mapsto X \star y \mapsto Y) \wedge X > Y)$. Simplicity for the logician is not necessarily simplicity for the programmer: we don't want to make specifications so bloated that nobody can bear to write or read them. Maybe one day we will have to embrace the complexities of this approach, but surely not before we've tried to find something simpler.

The next most obvious approach is to use an ownership predicate to describe permission to use the stack, and employ it alongside remarks about the value of $x$.[2] If we make that predicate immune to substitution (easy enough: substitution already respects bound variable restrictions, for example) then we might plausibly write

$$\begin{array}{l} \{(\mathrm{Own}(x) \star \mathrm{Own}(y)) \wedge x + 1 = 2y\} \\ \quad x := x + 1 \\ \{(\mathrm{Own}(x) \star \mathrm{Own}(y)) \wedge x = 2y\} \end{array} \tag{11}$$

and

$$\{(\mathrm{Own}(x) \star \mathrm{Own}(y)) \wedge y = 3\}\, x := y\, \{(\mathrm{Own}(x) \star \mathrm{Own}(y)) \wedge x = 3\} \tag{12}$$

[2] This was suggested to me at various time and by various people — Jules Bean and Hongseok Yang at least. Foolishly, I didn't listen to any of them. Instead I worked backwards from a plausible formalism and only at last realised that I'd arrived at what they had already proposed.

These examples do obey Hoare's assignment axiom (7) if you allow that $\mathrm{Own}(x)^{x}_{E}$ is $\mathrm{Own}(x)$. That reading of substitution is necessary just because assignment doesn't affect ownership (see section 2.1).

$\mathrm{Own}(x)$ is a predicate which is true iff the stack consists of a single variable called $x$ which you're allowed to read and write. It follows that $\mathrm{Own}(x) \star \mathrm{Own}(y)$ puts two single-variable stacks together to make a two-variable stack, that $\mathrm{Own}(x) \star \mathrm{Own}(x)$ should be false, that $\mathrm{Own}(x) \wedge \mathrm{Own}(y)$ should mean that $x$ and $y$ name the same variable (that is, $x$ and $y$ are aliases), that $\mathrm{Own}(x) \vee \mathrm{Own}(y)$ means I'm not sure what the variable's name is (not the same thing as an alias!), and that $\neg(\mathrm{Own}(x))$ means a stack which is anything but a singleton containing $x$. This makes the stack claim $\mathrm{Own}(x) \wedge x = 3$ closely analagous with the heap claim $10 \mapsto 3$.

I can deal with partial ownership using the ideas that lie behind heap permissions [4]. A fractional permission[3] or a counted read permission allows reading but not writing; a source permission allows creation of counted read permissions. $\mathrm{Own}_{0.5}(x)$ is a fractional read permission; $\mathrm{Own}^{>}(x)$ is a counted read permission; $\mathrm{Own}^{n}(x)$ is a source permission from which $n$ counted read permissions have been extracted; undecorated $\mathrm{Own}(x)$ is equivalent to $\mathrm{Own}_{1.0}(x)$ or $\mathrm{Own}^{0}(x)$ — all by syntactic analogy with the heap permissions $E \xmapsto{0.5} \_$, $E \rightarrowtail \_$ and $E \overset{n}{\mapsto} \_$. Partial ownership combines in the obvious way — for example $\mathrm{Own}_{z}(x) \star \mathrm{Own}_{z'}(x) \iff \mathrm{Own}_{z+z'}(x) \wedge z > 0 \wedge z' > 0$ — and it allows certain deductions — for example $\mathrm{Own}^{E}(x) \Rightarrow E \geq 0$. Substitution in $\mathrm{Own}^{E}(x)$ can affect $E$ but not $x$.

You might reasonably expect that the model of this logic would be separated stacks defined by the Own predicate. It is, more or less, but the principle that assignment doesn't affect ownership means that we need something more. The model, given to me by Hongseok Yang, is a set $O$ of owned variables and a stack $s$ mapping variables to integers. The stack isn't split by $\star$, but $O$ is:

$$(O, s) \star (O', s') = \text{if } O \cap O' = \emptyset \ \wedge \ s = s' \text{ then } (O \cup O', s) \text{ else undefined} \quad (13)$$

$$\begin{array}{lll}
(O, s) \vDash \mathrm{Own}(x) & \text{iff } x \in O \\
(O, s) \vDash P1 \star P2 & \text{iff} \left( \begin{array}{l} O1, O2 \text{ exist } s.t. \\ (O1, s) \star (O2, s) = (O, s) \text{ and} \\ (O1, s) \vDash P1 \text{ and } (O2, s) \vDash P2 \end{array} \right) \\
(O, s) \vDash P1 \wedge P2 & \text{iff } (O, s) \vDash P1 \text{ and } (O, s) \vDash P2 \\
(O, s) \vDash P1 \vee P2 & \text{iff } (O, s) \vDash P1 \text{ or } (O, s) \vDash P2 \\
(O, s) \vDash \neg P & \text{iff not } ((O, s) \vDash P)
\end{array} \quad (14)$$

[3] I know already that fractions are not quite the thing because they don't allow straightforward inductive definitions of fractional access to data structures (see [4] for details). I already know how to fix the problem, but fractions won't lead us astray in the examples in this paper.

The model allows us to judge that a particular assertion is *well-supported*:

**Definition 3.1** $P$ is well-supported iff when $(O, s) \models P$ then, for any stack $s'$ s.t. for all $o$ in $O$ $s(o) = s'(o)$, $(O, s') \models P$.

This allows $P$ to mention variables mapped by $s$ but outside $O$, so long as the particular values of those variables doesn't matter. That allows me to say that $\mathrm{Own}(x)$ is entirely equivalent to $\mathrm{Own}(x) \wedge y = y$, and that both are well-supported. Similarly, $(\mathrm{Own}(x) \wedge y = 3) \star (\mathrm{Own}(y) \wedge y = 3)$ is well-supported and equivalent to $\mathrm{Own}(x) \star \mathrm{Own}(y) \wedge y = 3$. Both these properties simplify the treatment of variable assignment below.

The logic is restricted to deductions about well-supported assertions, but I've introduced a special notation of *variable contexts*. To simplify the discussion I've begun by treating a language which doesn't use the heap, as in conventional Hoare logic. This feels more than a little cowardly, and to recover my pride I've pointed out at the end of this paper how easy it is to deal with the heap as well.

## 4 Variable contexts

To simplify assertions, substitution and the notion of *well-scoping* I introduce some notation.

$$P ::= \Gamma \models Q \mid P \star P \mid P \wedge P \mid P \vee P \tag{15}$$

where the *variable context* $\Gamma$ is a comma-separated list of variable names (possibly decorated with permissions) and $Q$ is a conventional predicate logic assertion (nothing about the heap: no $\mapsto$, no **emp**, no $\star$, no $-\!\!\star$).

A *contextual* assertion $\mathit{x1}, \mathit{x2}, \ldots, \mathit{xn} \models Q$ is shorthand for $(\mathrm{Own}(\mathit{x1}) \star \mathrm{Own}(\mathit{x2}) \star \cdots \star \mathrm{Own}(\mathit{xn})) \wedge Q$. It asserts ownership of all and only the variables in the context and the truth of $Q$. Comma mimics $\star$, so it's associative and commutative. A contextual assertion is *well-scoped* if all the free variables of the formula are owned in the context or if it can be transformed, by equivalence transformations, into a well-scoped assertion. Clearly, well-scoped assertions are well-supported.

A contextual assertion is false if the variable context claims a variable more than once (just because $\mathrm{Own}(x) \star \mathrm{Own}(x)$ is false).

An unsupported assertion is not necessarily false, but since it isn't properly resourced it can't be used: all my rules demand well-supported assertions.

$\mathit{P1} \star \mathit{P2}$, where $\mathit{P1}$ and $\mathit{P2}$ are contextual assertions, splits the stack. Because you can always use fractional permissions[4] you can always split a contextual assertion. You can always join them too: $(\Gamma \models Q) \star (\Delta \models R)$ is equivalent to $\Gamma, \Delta \models Q \wedge R$. In particular that means it's false if $\Gamma, \Delta$ claims the

[4] [4] says you can't split counted read permissions or source permissions, but I now know how to do that too. It would be a distraction to give details here.

same variable more than once. You can also do obvious stuff like transforming $P1 \star (P2 \vee P3)$ into $(P1 \star P2) \vee (P1 \star P3)$.

It's clear what $\neg(\Gamma \vDash Q)$ means – the stack is such that $\Gamma \vDash Q$ is false – but I shan't make any use of it.

### *4.1 Semaphore permissions*

In hardware terms a semaphore is just a variable. In software terms it's something we can P and V from outside, and treat as a read/write variable inside its CCR-body (see (4)). That's an obvious case for permissions. In the examples of this paper I don't really need to account for semaphore permissions, but I show them anyway because eventually it will be necessary.

The total permission for a semaphore $m$ splits into three parts: $m_{\mathrm{P}}$, $m_{\mathrm{V}}$ and $m_{\mathrm{S}}$. The $_{\mathrm{S}}$ permission is held by the semaphore resource-bundle itself, and allows reading and writing. The $_{\mathrm{P}}$ and $_{\mathrm{V}}$ permissions are existence permissions, and give no access to the variable at all, but they do allow P and V respectively. $_{\mathrm{P}}$ and $_{\mathrm{V}}$ are infinitely divisible, to allow arbitrary sharing of the semaphore, but I shall spare you that notational complexity in this paper: in my examples I will pretend, for simplicity, that each semaphore has only one user.

## 5 Rules

In deductions I permit only well-scoped assertions, and that restriction replaces previous side conditions restricting use of variables.

The frame rule is now almost unadorned, but it does require a side condition (which you can't deduce from the general requirement that $P \star Q$ and $P \star R$ are well-scoped):

$$\frac{\{Q\}\,C\,\{R\}}{\{P \star Q\}\,C\,\{P \star R\}}\ \text{, provided } P \text{ is well-scoped} \tag{16}$$

The side condition that $C$ doesn't overwrite free variables of $P$ has become a side condition of well-scoping, and that is almost a syntactic condition.[5] The effect, because the variable-assignment rule (19) insists that a variable which is overwritten must be totally owned, is what we need: $C$ can't overwrite the free variables of $P$ because the context of $P$ must claim at least partial ownership, and then $\star$-separation in $P \star Q$ means that $Q$ can't have total ownership.

The concurrency rule is unchanged from (2) but now needs no side condition because the assertions must be well-scoped, therefore well-supported and stack-separated.

[5] Any sensible tool-writer would surely insist that $P$ must be syntactically well-scoped, with no need for equivalence transformations.

The CCR rule now combines stacks on entry to the command $C$ and splits them on exit. It needs an additional antecedent because there must be (not necessarily total) permission to read the resource bundle $b$. Well-scoping of $b$'s invariant $I_b$ is a side condition: once again, this is almost syntactic, and will be in any case required directly by the rule (not shown here) which deals with the declaration of $b$.

$$\frac{Q \rightarrow (\mathrm{Own}_{\_}(b) \star \mathrm{true}) \quad \{(Q \star I_b) \wedge G\}\, C\, \{R \star I_b\}}{\{Q\}\ \text{with}\ b\ \text{when}\ G\ \text{do}\ C\ \text{od}\ \{R\}}\ \text{, provided } I_b \text{ is well-scoped} \tag{17}$$

The antecedent precondition $(Q \star I_b) \wedge G$ gives a little trouble. $G$, a program formula, can't have a context, and so variables in $G$ are outside the scope of any context in $(Q \star I_b) \wedge G$. We have to read it as shorthand for an assertion (not necessarily a well-scoped assertion) which combines the stacks of $Q$ and $I_b$ and distributes $G$ across the result.[6] For example:

$$\begin{aligned} Q&: m_{\mathrm{P}}, t^{>} \vDash \mathrm{true} \\ I_m&: (m_{\mathrm{S}} \vDash m = 0) \vee (m_{\mathrm{S}}, c, t^{c} \vDash c \geq 0) \\ G&: m = 1 \\ (Q \star I_m) \wedge G&: \begin{array}{l} (m_{\mathrm{P}}, t^{>}, m_{\mathrm{S}} \vDash m = 0 \wedge m = 1) \vee \\ (m_{\mathrm{P}}, t^{>}, m_{\mathrm{S}}, c, t^{c} \vDash c \geq 0 \wedge m = 1) \end{array} \end{aligned} \tag{18}$$

The variable-assignment axiom becomes a rule whose antecedent is even more almostly syntactic than the side conditions above:

$$\frac{R \rightarrow (\mathrm{Own}(x) \star \mathrm{true})}{\{R^{x}_{E}\}\, x := E\, \{R\}} \tag{19}$$

There is no aliasing side condition (no need to hide behind the restrictions of separation logic: see section 6.1), but of course $R$ and $R^{x}_{E}$ must be well-scoped. There are some interesting consequences of this definition.

(i) Splitting the stack with $(\star)$ and then applying variable assignment is quite ok:

$$\{(x \vDash y = 3) \star (y \vDash y = 3)\}\, x := y\, \{(x \vDash x = 3) \star (y \vDash y = 3)\} \tag{20}$$

The precondition looks odd, because $x \vDash y = 3$ is ill-scoped. If we combine the stacks in the postcondition and then apply the axiom

$$\{x, y \vDash y = 3 \wedge y = 3\}\, x := y\, \{x, y \vDash x = 3 \wedge y = 3\} \tag{21}$$

nothing looks out of place. In fact there's nothing wrong with the split-stack deduction (20) either: $(x \vDash y = 3) \star (y \vDash y = 3)$ is altogether

[6] I believe that's what Yang's model says it means, but I explained it in detail in case you find it as surprising as I did.

well-supported, and well-scoped too because it's easily converted to the unfrightening $x, y \vDash y = 3$.

(ii) $R$ and $R^x_E$ must be well-scoped, and both must include $\mathrm{Own}(x)$, but otherwise $x$ need not occur in $R$. This allows surprises like

$$\{x \vDash \text{true}\}\, x := y + 1\, \{x \vDash \text{true}\} \tag{22}$$

There's no race if you take no notice! I'm rather fond of that mischievous possibility. But it, too, can give a surprise:

$$\{x \vDash y + 1 = y + 1\}\, x := y + 1\, \{x \vDash x = x\} \tag{23}$$

The precondition *is* well-supported, and it's well-scoped because it easily converts to $x \vDash$ true.

## 6 Rules in action

Because Hoare logic has nothing to say about resource, we can make misleading deductions like

$$\{y = 1\}\, x := 7\, \{y = 1\} \tag{24}$$

which weasel out of their obligation to display all the effects of an assigment. In separation logic the frame rule catches the weasel, because the side condition disallows the deduction

$$\frac{\{y = 1\}\, x := 7\, \{y = 1\}}{\{x = 3 \star y = 1\}\, x := 7\, \{x = 3 \star y = 1\}} \tag{25}$$

since $x := 7$ alters a variable free in $x = 3$.

Variable contexts catch the varmint too, in either of two traps. An antecedent which doesn't own $x$

$$\frac{\{y \vDash y = 1\}\, x := 7\, \{y \vDash y = 1\}}{\{(x \vDash x = 3) \star (y \vDash y = 1)\}\, x := 7\, \{(x \vDash x = 3) \star (y \vDash y = 1)\}} \tag{26}$$

fails because it doesn't have the resource to support the assignment. An antecedent which claims $x$

$$\frac{\{x, y \vDash y = 1\}\, x := 7\, \{x, y \vDash y = 1\}}{\{(x \vDash x = 3) \star (x, y \vDash y = 1)\}\, x := 7\, \{(x \vDash x = 3) \star (x, y \vDash y = 1)\}} \tag{27}$$

can't at the same time leave it behind: the consequent pre-condition is false.

### *6.1 Variable aliasing and contexts*

In the variable-assignment rule (19) we must own the assigned variable $x$ and any variables mentioned in $R$ or in $R^x_E$ must be named in the context. Since

$$
\begin{array}{l}
\{m_{\mathrm{PV}} \vDash \mathrm{true}\} \\
\mathrm{P}(m) : \left(\begin{array}{l}
\left\{\left((m_{\mathrm{PV}} \vDash \mathrm{true}) \star \left((m_{\mathrm{S}} \vDash m = 0) \vee (m_{\mathrm{S}}, c \vDash m = 1 \wedge c \geq 0)\right)\right) \wedge m = 1\right\} \therefore \\
\left\{\begin{array}{l} (m_{\mathrm{PV}}, m_{\mathrm{S}} \vDash \mathrm{true} \wedge m = 0 \wedge m = 1) \vee \\ (m_{\mathrm{PV}}, m_{\mathrm{S}}, c \vDash \mathrm{true} \wedge m = 1 \wedge c \geq 0 \wedge m = 1) \end{array}\right\} \therefore \\
\{m_{\mathrm{PV}}, m_{\mathrm{S}}, c \vDash m = 1 \wedge c \geq 0\} \\
\quad m := 0 \\
\{m_{\mathrm{PV}}, m_{\mathrm{S}}, c \vDash m = 0 \wedge c \geq 0\} \therefore \\
\{(m_{\mathrm{PV}}, c \vDash c \geq 0) \star (m_{\mathrm{S}} \vDash m = 0)\} \therefore \\
\left\{(m_{\mathrm{PV}}, c \vDash c \geq 0) \star \left((m_{\mathrm{S}} \vDash m = 0) \vee (m_{\mathrm{S}}, c \vDash m = 1 \wedge c \geq 0)\right)\right\}
\end{array}\right) \\
\{m_{\mathrm{PV}}, c \vDash c \geq 0\} \\
\quad c{+}{+} \\
\{m_{\mathrm{PV}}, c \vDash c > 0\} \\
\mathrm{V}(m) : \left(\begin{array}{l}
\left\{\left((m_{\mathrm{PV}}, c \vDash c > 0) \star \left((m_{\mathrm{S}} \vDash m = 0) \vee (m_{\mathrm{S}}, c \vDash m = 1 \wedge c \geq 0)\right)\right) \wedge \mathrm{true}\right\} \therefore \\
\left\{\begin{array}{l} (m_{\mathrm{PV}}, c, m_{\mathrm{S}} \vDash c > 0 \wedge m = 0 \wedge \mathrm{true}) \vee \\ (m_{\mathrm{PV}}, c, m_{\mathrm{S}}, c \vDash c > 0 \wedge m = 1 \wedge c \geq 0 \wedge \mathrm{true}) \end{array}\right\} \therefore \\
\{m_{\mathrm{PV}}, c, m_{\mathrm{S}} \vDash c > 0 \wedge m = 0\} \\
\quad m := 1 \\
\{m_{\mathrm{PV}}, c, m_{\mathrm{S}} \vDash c > 0 \wedge m = 1\} \therefore \\
\{(m_{\mathrm{PV}} \vDash \mathrm{true}) \star (m_{\mathrm{S}}, c \vDash m = 1 \wedge c \geq 0)\} \therefore \\
\left\{(m_{\mathrm{PV}} \vDash \mathrm{true}) \star \left((m_{\mathrm{S}} \vDash m = 0) \vee (m_{\mathrm{S}}, c \vDash m = 1 \wedge c \geq 0)\right)\right\}
\end{array}\right) \\
\{m_{\mathrm{PV}} \vDash \mathrm{true}\}
\end{array}
$$

Fig. 1. Variable-ownership transfer from and to a mutex

the context describes a $\star$-separated list of Own predicates, that's it for aliasing: there can't be any aliases for $x$ around to upset the substitution.

But I might do even better and control aliasing. $\mathrm{Own}(x) \wedge \mathrm{Own}(y)$ describes a stack with two names for the same cell. In bold analogy with BI, I might use comma for separation, and semicolon for identity of variables. Substitution would have to be context-sensitive, which isn't too hard (I've been doing worse for years in Jape [1], and that runs at a reasonable speed):

$$
\begin{array}{rcl}
\Gamma, x, y \vDash y^x_E & \Rightarrow & y \\
\Gamma, x; y \vDash y^x_E & \Rightarrow & E
\end{array}
\tag{28}
$$

Taraa, ching! as we used to say in the 1960s. To avoid muddying the waters I won't try to develop this idea further for the time being (I think it would need a well-scoped $R$ at least, and no doubt more).

It's nice that the frame rule works without a significant side condition and that variable aliasing is under control, but none of this is very surprising or particularly useful. The point of introducing variable contexts is to explain variable-ownership transfer using CCRs.

$$\begin{array}{c|c|c|c}
\mathrm{P}(m);\ c{+}{+};\ \mathrm{V}(m) & \mathrm{P}(m);\ c{+}{+};\ \mathrm{V}(m) & & \mathrm{P}(m);\ c{+}{+};\ \mathrm{V}(m) \\
\ldots & \ldots & \ldots & \ldots \\
\mathrm{P}(m);\ c{-}{-};\ \mathrm{V}(m) & \mathrm{P}(m);\ c{-}{-};\ \mathrm{V}(m) & & \mathrm{P}(m);\ c{-}{-};\ \mathrm{V}(m)
\end{array}$$

Fig. 2. Is $c \geq 0$ invariant?

## 7 Mutexes and variable-ownership transfer

Consider the program fragment

$$\mathrm{P}(m);\ c{+}{+};\ \mathrm{V}(m) \tag{29}$$

The semaphore $m$ is a mutex (a binary semaphore, one in which $m = 0 \vee m = 1$) which owns/contains an integer variable $c$ and which protects a critical section, in this case the single instruction $c$++. I'd like to prove that this fragment maintains a semaphore invariant which includes $c \geq 0$.

The program takes $c$ out of the semaphore with P($m$), increments it, and puts it back with V($m$). By analogy with O'Hearn's buffer-ownership-transfer example [15] the invariant must surely describe a disjunction of states. In the normal or 'checked-in' state outside the critical section it must be $m_{\mathrm{S}}, c \vDash m = 1 \wedge c \geq 0$; in the 'checked-out' state within a critical section we need $m_{\mathrm{S}} \vDash m = 0$. The invariant is therefore

$$(m_{\mathrm{S}} \vDash m = 0) \vee (m_{\mathrm{S}}, c \vDash m = 1 \wedge c \geq 0) \tag{30}$$

Proof that (29) preserves the semaphore invariant is shown in figure 1. The only tricky bit is the simplification on entry to V($m$). The second disjunct in the second line is false because it claims $c$ twice. This is the separation property at work: *if* every thread plays by the rules, and *if* a thread has the $c$ permission, *then* the semaphore must be in the $m_{\mathrm{S}} \vDash m = 0$ state.

The critical section ($c$++) is mutually exclusive with any other critical section using the same mutex because the program gets exclusive permission to access $c$. It's impossible, by the separation property, that any other thread could have the same permission at the same time.

Even this simple example needs a lot of tedious calculation to prove a simple ownership transfer – "a great deal of work for such a simple matter" as Burstall [8] put it. At any rate, in other examples I'll omit the rearrangement on entry and the consequence step on exit from a semaphore (the first and last lines in P($m$) and V($m$) in figure 1).

## 8 Auxiliary ticket transfer

The readers-and-writers example (figure 4) presents a particular permission-accounting difficulty. The reader mutex contains various resources, amongst which is a count variable $c$; we want to guarantee that $c \geq 0$ is part of the semaphore invariant. The count variable is incremented in the reader prologue

$$
\begin{array}{l}
\{m_{\mathrm{PV}} \vDash \mathrm{true}\} \\
\quad \mathrm{P}(m) \\
\{m_{\mathrm{PV}}, c, t^{c} \vDash c \geq 0\} \\
\quad c{+}{+} \\
\{m_{\mathrm{PV}}, c, t^{c-1} \vDash c > 0\} \therefore \{m_{\mathrm{PV}}, c, t^{c}, t^{>} \vDash c > 0\} \\
\quad \mathrm{V}(m) \\
\{m_{\mathrm{PV}}, t^{>} \vDash \mathrm{true}\} \\
\ldots \\
\{m_{\mathrm{PV}}, t^{>} \vDash \mathrm{true}\} \\
\quad \mathrm{P}(m) : \left(\begin{array}{l}
\left\{\begin{array}{l}
(m_{\mathrm{PV}}, t^{>}, m_{\mathrm{S}} \vDash \mathrm{true} \wedge m = 0 \wedge m = 1) \vee \\
(m_{\mathrm{PV}}, t^{>}, m_{\mathrm{S}}, c, t^{c} \vDash \mathrm{true} \wedge m = 1 \wedge c \geq 0 \wedge m = 1)
\end{array}\right\} \therefore \\
\{m_{\mathrm{PV}}, m_{\mathrm{S}}, c, t^{c-1} \vDash m = 1 \wedge c \geq 0\} \;\therefore \\
\{m_{\mathrm{PV}}, m_{\mathrm{S}}, c, t^{c-1} \vDash m = 1 \wedge c \geq 0 \wedge c - 1 \geq 0\} \;\therefore \\
\{m_{\mathrm{PV}}, m_{\mathrm{S}}, c, t^{c-1} \vDash m = 1 \wedge c > 0\} \\
\quad m := 0 \\
\{m_{\mathrm{PV}}, m_{\mathrm{S}}, c, t^{c-1} \vDash m = 0 \wedge c > 0\} \;\therefore \\
\{(m_{\mathrm{PV}}, c, t^{c-1} \vDash c > 0) \star (m_{\mathrm{S}} \vDash m = 0)\}
\end{array}\right) \\
\{m_{\mathrm{PV}}, c, t^{c-1} \vDash c > 0\} \\
\quad c{-}{-} \\
\{m_{\mathrm{PV}}, c, t^{c} \vDash c \geq 0\} \\
\quad \mathrm{V}(m) \\
\{m_{\mathrm{PV}} \vDash \mathrm{true}\}
\end{array}
$$

Fig. 3. Ticket permits variable decrement to preserve invariant

and can be treated logically just as in figure 1. But in the epilogue the count is decreased: if all we can rely on is $c \geq 0$ from the invariant, then $c{-}{-}$ can make $c$ negative, and the invariant is destroyed.

The simplest way to show the problem is to compose lots of copies of a program which first increments and then decrements a count, each time in a critical section, shown in figure 2. It's completely obvious that this program does exactly as many decrements as increments, and will preserve $c \geq 0$ in the semaphore invariant. The problem is to prove it by local reasoning. The solution depends on the use of an auxiliary $t$ to track the number of increments and decrements that have occurred, *using permissions to record the difference.*[7] Recall that, by analogy with heap permissions, $t^{N}$ is a counting source permission and $t^{>}$ a counted read permission: then the semaphore invariant is

$$(m_{\mathrm{S}} \vDash m = 0) \vee (m_{\mathrm{S}}, c, t^{c} \vDash m = 1 \wedge c \geq 0) \tag{31}$$

I don't really need $c \geq 0$ in the second disjunct because $t^{c}$ in the context implies it — recall the treatment of heap permissions in [4] — but I've left it in place for clarity.

The proof is now straightforward and shown in figure 3. I've skipped most

[7] Matthew Parkinson and I independently devised solutions using read permission on an auxiliary heap buffer as a ticket. Cristiano Calcagno pointed out that permission to access a variable alone would do.

```
READERS                                                   WRITER

prologue: ( P(m);
            c++;
            if c = 1 then P(w) else skip fi;
            V(m); )
                                                          P(w);
   .....                                                     .....
reading happens                                           writing happens
   .....                                                     .....
                                                          V(w);
epilogue: ( P(m);
            c−−;
            if c = 0 then V(w) else skip fi;
            V(m) )
```

Fig. 4. Readers and writers (from [9], with shortened names and explicit skips)

of the internal semaphore deductions, because they are so similar to those in figure 1. Note that the program emerges from the first P with a source permission $t^c$; incrementing $c$ means that it can put back $t^c$ in the following V and still be left with $t^>$. Note also the pre- and post-conditions of $c$++: substitution does affect source-permission formulae in the context.

The action is all in the second P. The first deduction is standard, selecting one of the invariant alternatives based on the value of $m$. But when we combine $t^c$ from the semaphore with $t^>$ from the thread, we get $t^{c-1}$ and from that, by the properties of counting permissions, we can deduce $c-1 \geq 0$ and therefore $c > 0$. Once again the separation property is doing the work: if we really have a read permission and the semaphore really has the source permission, then the re-combination will be another valid source permission.

Because of the way that counting permissions work, possession of the read permission $t^>$ is a guarantee that somewhere there is a source permission $t^N$ with $N \geq 1$. In this program we know where it is: locked in the mutex, or checked out in a critical section. The ticket-variable $t$ is a perfect auxiliary. The program knows nothing about it, because we haven't had to insert any instructions which manipulate it.

Shazam! — it works, with local reasoning, the separation property, and a semaphore invariant.

I can't yet prove that if the program in figure 2 starts with $c = C$ then it finishes with $c = C$.

## 9 Readers and writers

Enough preparation: I am now set up to assault the citadel of the readers-and-writers example, shown in more or less its original form in figure 4. The semaphore $m$ (*mutex* in the original) holds a variable $c$ (*readcount* in the

$$\{m_{\mathrm{PV}} \vDash \mathrm{true}\}$$

$$\mathrm{P}(m) : \left(\begin{array}{l}
\left\{\begin{array}{l}
(m_{\mathrm{PV}}, m_{\mathrm{S}} \vDash \mathrm{true} \wedge m = 0 \wedge m = 1) \vee \\
(m_{\mathrm{PV}}, m_{\mathrm{S}}, w_{\mathrm{PV}}, c, t \vDash \mathrm{true} \wedge m = 1 \wedge c = 0 \wedge m = 1) \vee \\
(m_{\mathrm{PV}}, m_{\mathrm{S}}, w_{\mathrm{PV}}, c, t^c, y^c \vDash \mathrm{true} \wedge m = 1 \wedge c > 0 \wedge m = 1)
\end{array}\right\} \therefore \\
\left\{(m_{\mathrm{PV}}, m_{\mathrm{S}}, w_{\mathrm{PV}}, c, t \vDash c = 0 \wedge m = 1) \vee (m_{\mathrm{PV}}, m_{\mathrm{S}}, w_{\mathrm{PV}}, c, t^c, y^c \vDash c > 0 \wedge m = 1)\right\} \\
\quad m := 0 \\
\left\{(m_{\mathrm{PV}}, m_{\mathrm{S}}, w_{\mathrm{PV}}, c, t \vDash c = 0 \wedge m = 0) \vee (m_{\mathrm{PV}}, m_{\mathrm{S}}, w_{\mathrm{PV}}, c, t^c, y^c \vDash c > 0 \wedge m = 0)\right\} \therefore \\
\left\{\left(\begin{array}{l}
(m_{\mathrm{PV}}, w_{\mathrm{PV}}, c, t \vDash c = 0) \vee \\
(m_{\mathrm{PV}}, w_{\mathrm{PV}}, c, t^c, y^c \vDash c > 0)
\end{array}\right) \star (m_{\mathrm{S}} \vDash m = 0)\right\}
\end{array}\right)$$

$$\{(m_{\mathrm{PV}}, w_{\mathrm{PV}}, c, t \vDash c = 0) \vee (m_{\mathrm{PV}}, w_{\mathrm{PV}}, c, t^c, y^c \vDash c > 0)\}$$

$$\quad c{+}{+}$$

$$\{(m_{\mathrm{PV}}, w_{\mathrm{PV}}, c, t \vDash c - 1 = 0) \vee (m_{\mathrm{PV}}, w_{\mathrm{PV}}, c, t^{c-1}, y^{c-1} \vDash c - 1 > 0)\}$$

$$\quad \mathrm{if}\ c = 1\ \mathrm{then}$$

$$\qquad \{(m_{\mathrm{PV}}, w_{\mathrm{PV}}, c, t \vDash c - 1 = 0 \wedge c = 1) \vee (m_{\mathrm{PV}}, w_{\mathrm{PV}}, c, t^{c-1}, y^{c-1} \vDash c - 1 > 0 \wedge c = 1)\} \therefore$$

$$\qquad \{m_{\mathrm{PV}}, w_{\mathrm{PV}}, c, t \vDash c = 1\}$$

$$\qquad \mathrm{P}(w) : \left(\begin{array}{l}
\left\{\begin{array}{l}
(m_{\mathrm{PV}}, w_{\mathrm{PV}}, c, t, w_{\mathrm{S}}, y \vDash c = 1 \wedge w = 1 \wedge w = 1) \vee \\
(m_{\mathrm{PV}}, w_{\mathrm{PV}}, c, t, w_{\mathrm{S}} \vDash c = 1 \wedge w = 0 \wedge w = 1)
\end{array}\right\} \therefore \\
\{m_{\mathrm{PV}}, w_{\mathrm{PV}}, c, t, w_{\mathrm{S}}, y \vDash c = 1 \wedge w = 1\} \\
\quad w := 0 \\
\{m_{\mathrm{PV}}, w_{\mathrm{PV}}, c, t, w_{\mathrm{S}}, y \vDash c = 1 \wedge w = 0\} \therefore \\
\{(m_{\mathrm{PV}}, w_{\mathrm{PV}}, c, t, y \vDash c = 1) \star (w_{\mathrm{S}} \vDash w = 0)\}
\end{array}\right)$$

$$\qquad \{m_{\mathrm{PV}}, w_{\mathrm{PV}}, c, t, y \vDash c = 1\} \therefore$$

$$\qquad \{m_{\mathrm{PV}}, w_{\mathrm{PV}}, c, t^{c-1}, y^{c-1} \vDash c = 1\}$$

$$\quad \mathrm{else}$$

$$\qquad \{(m_{\mathrm{PV}}, w_{\mathrm{PV}}, c, t \vDash c - 1 = 0 \wedge c \neq 1) \vee (m_{\mathrm{PV}}, w_{\mathrm{PV}}, c, t^{c-1}, y^{c-1} \vDash c - 1 > 0 \wedge c \neq 1)\} \therefore$$

$$\qquad \{m_{\mathrm{PV}}, w_{\mathrm{PV}}, c, t^{c-1}, y^{c-1} \vDash c > 1)\}$$

$$\qquad\quad \mathrm{skip}$$

$$\qquad \{m_{\mathrm{PV}}, w_{\mathrm{PV}}, c, t^{c-1}, y^{c-1} \vDash c > 1)\}$$

$$\quad \mathrm{fi}$$

$$\{m_{\mathrm{PV}}, w_{\mathrm{PV}}, c, t^{c-1}, y^{c-1} \vDash c \geq 1)\} \therefore$$

$$\{m_{\mathrm{PV}}, w_{\mathrm{PV}}, c, t^c, t^>, y^c, y^> \vDash c \geq 1)\}$$

$$\mathrm{V}(m) : \left(\begin{array}{l}
\left\{\begin{array}{l}
(m_{\mathrm{PV}}, w_{\mathrm{PV}}, c, t^c, t^>, y^c, y^>, m_{\mathrm{S}} \vDash c \geq 1 \wedge m = 0 \wedge \mathrm{true}) \vee \\
(m_{\mathrm{PV}}, w_{\mathrm{PV}}, c, t^c, t^>, y^c, y^>, m_{\mathrm{S}}, w_{\mathrm{PV}}, c, t \vDash c \geq 1 \wedge m = 1 \wedge c = 0 \wedge \mathrm{true}) \vee \\
(m_{\mathrm{PV}}, w_{\mathrm{PV}}, c, t^c, t^>, y^c, y^>, m_{\mathrm{S}}, w_{\mathrm{PV}}, c, t^c, y^c \vDash c \geq 1 \wedge m = 1 \wedge c > 0 \wedge \mathrm{true})
\end{array}\right\} \therefore \\
\{m_{\mathrm{PV}}, w_{\mathrm{PV}}, c, t^c, t^>, y^c, y^>, m_{\mathrm{S}} \vDash c \geq 1 \wedge m = 0\} \\
\quad m := 1 \\
\{m_{\mathrm{PV}}, w_{\mathrm{PV}}, c, t^c, t^>, y^c, y^>, m_{\mathrm{S}} \vDash c \geq 1 \wedge m = 1\} \therefore \\
\left\{(m_{\mathrm{PV}}, t^>, y^> \vDash \mathrm{true}) \star (m_{\mathrm{S}}, w_{\mathrm{PV}}, c, t^c, y^c \vDash m = 1 \wedge c > 0)\right\}
\end{array}\right)$$

$$\{m_{\mathrm{PV}}, t^>, y^> \vDash \mathrm{true}\}$$

Fig. 5. Proof of the reader prologue

original) and, when $c > 0$, it also holds a source permission to a shared resource which can be read in the 'reading happens' section between prologue and epilogue. The semaphore $w$ (*write* in the original) holds a total permission to the same shared resource.

$$
\begin{array}{l}
\{m_{\mathrm{PV}}, y^{>}, t^{>} \vDash \text{true}\} \\
\mathrm{P}(m) : \left(\begin{array}{l}
\left\{\begin{array}{l}
(m_{\mathrm{PV}}, y^{>}, t^{>}, m_{\mathrm{S}} \vDash \text{true} \wedge m = 0 \wedge m = 1) \vee \\
(m_{\mathrm{PV}}, y^{>}, t^{>}, m_{\mathrm{S}}, w_{\mathrm{PV}}, c, t \vDash \text{true} \wedge m = 1 \wedge c = 0 \wedge m = 1) \vee \\
(m_{\mathrm{PV}}, y^{>}, t^{>}, m_{\mathrm{S}}, w_{\mathrm{PV}}, c, t^{c}, y^{c} \vDash \text{true} \wedge m = 1 \wedge c > 0 \wedge m = 1)
\end{array}\right\} \therefore \\
\{m_{\mathrm{PV}}, y^{>}, t^{>}, m_{\mathrm{S}}, w_{\mathrm{PV}}, c, t^{c}, y^{c} \vDash m = 1 \wedge c > 0\} \therefore \\
\{m_{\mathrm{PV}}, m_{\mathrm{S}}, w_{\mathrm{PV}}, c, t^{c-1}, y^{c-1} \vDash m = 1 \wedge c > 0\} \\
\quad m := 0 \\
\{m_{\mathrm{PV}}, m_{\mathrm{S}}, w_{\mathrm{PV}}, c, t^{c-1}, y^{c-1} \vDash m = 0 \wedge c > 0\} \therefore \\
\{(m_{\mathrm{PV}}, w_{\mathrm{PV}}, c, t^{c-1}, y^{c-1} \vDash c > 0) \star (m_{\mathrm{S}} \vDash m = 0)\}
\end{array}\right) \\
\{m_{\mathrm{PV}}, w_{\mathrm{PV}}, c, t^{c-1}, y^{c-1} \vDash c > 0\} \\
\quad c{-}{-} \\
\{m_{\mathrm{PV}}, w_{\mathrm{PV}}, c, t^{c}, y^{c} \vDash c \geq 0\} \\
\quad \text{if } c = 0 \text{ then} \\
\qquad \{m_{\mathrm{PV}}, w_{\mathrm{PV}}, c, t^{c}, y^{c} \vDash c \geq 0 \wedge c = 0\} \therefore \{m_{\mathrm{PV}}, w_{\mathrm{PV}}, c, t, y \vDash c = 0\} \\
\qquad \mathrm{V}(w) : \left(\begin{array}{l}
\left\{\begin{array}{l}
(m_{\mathrm{PV}}, w_{\mathrm{PV}}, c, t, y, w_{\mathrm{S}}, y \vDash c = 0 \wedge w = 1 \wedge \text{true}) \vee \\
(m_{\mathrm{PV}}, w_{\mathrm{PV}}, c, t, y, w_{\mathrm{S}} \vDash c = 0 \wedge w = 0 \wedge \text{true})
\end{array}\right\} \therefore \\
\{m_{\mathrm{PV}}, w_{\mathrm{PV}}, c, t, y, w_{\mathrm{S}} \vDash c = 0 \wedge w = 0\} \\
\quad w := 1 \\
\{m_{\mathrm{PV}}, w_{\mathrm{PV}}, c, t, y, w_{\mathrm{S}} \vDash c = 0 \wedge w = 1\} \therefore \\
\{(m_{\mathrm{PV}}, w_{\mathrm{PV}}, c, t \vDash c = 0) \star (w_{\mathrm{S}}, y \vDash w = 1)\}
\end{array}\right) \\
\qquad \{m_{\mathrm{PV}}, w_{\mathrm{PV}}, c, t \vDash c = 0\} \\
\quad \text{else} \\
\qquad \{m_{\mathrm{PV}}, w_{\mathrm{PV}}, c, t^{c}, y^{c} \vDash c \geq 0 \wedge c \neq 0\} \therefore \{m_{\mathrm{PV}}, w_{\mathrm{PV}}, c, t^{c}, y^{c} \vDash c > 0\} \\
\qquad\quad \text{skip} \\
\qquad \{m_{\mathrm{PV}}, w_{\mathrm{PV}}, c, t^{c}, y^{c} \vDash c > 0\} \\
\quad \text{fi} \\
\{(m_{\mathrm{PV}}, w_{\mathrm{PV}}, c, t \vDash c = 0) \vee (m_{\mathrm{PV}}, w_{\mathrm{PV}}, c, t^{c}, y^{c} \vDash c > 0)\} \\
\mathrm{V}(m) : \left(\begin{array}{l}
\left\{\begin{array}{l}
(m_{\mathrm{PV}}, w_{\mathrm{PV}}, c, t, m_{\mathrm{S}} \vDash c = 0 \wedge m = 0 \wedge \text{true}) \vee \\
(m_{\mathrm{PV}}, w_{\mathrm{PV}}, c, t, m_{\mathrm{S}}, w_{\mathrm{PV}}, c, t \vDash c = 0 \wedge m = 1 \wedge c = 0 \wedge \text{true}) \vee \\
(m_{\mathrm{PV}}, w_{\mathrm{PV}}, c, t, m_{\mathrm{S}}, w_{\mathrm{PV}}, c, t^{c}, y^{c} \vDash c = 0 \wedge m = 1 \wedge c > 0 \wedge \text{true}) \vee \\
(m_{\mathrm{PV}}, w_{\mathrm{PV}}, c, t^{c}, y^{c}, m_{\mathrm{S}} \vDash c > 0 \wedge m = 0 \wedge \text{true}) \vee \\
(m_{\mathrm{PV}}, w_{\mathrm{PV}}, c, t^{c}, y^{c}, m_{\mathrm{S}}, w_{\mathrm{PV}}, c, t \vDash c > 0 \wedge m = 1 \wedge c = 0 \wedge \text{true}) \vee \\
(m_{\mathrm{PV}}, w_{\mathrm{PV}}, c, t^{c}, y^{c}, m_{\mathrm{S}}, w_{\mathrm{PV}}, c, t^{c}, y^{c} \vDash c > 0 \wedge m = 1 \wedge c > 0 \wedge \text{true})
\end{array}\right\} \therefore \\
\left\{\begin{array}{l}
(m_{\mathrm{PV}}, w_{\mathrm{PV}}, c, t, m_{\mathrm{S}} \vDash c = 0 \wedge m = 0 \wedge \text{true}) \vee \\
(m_{\mathrm{PV}}, w_{\mathrm{PV}}, c, t^{c}, y^{c}, m_{\mathrm{S}} \vDash c > 0 \wedge m = 0 \wedge \text{true})
\end{array}\right\} \\
\quad m := 1 \\
\left\{\begin{array}{l}
(m_{\mathrm{PV}}, w_{\mathrm{PV}}, c, t, m_{\mathrm{S}} \vDash c = 0 \wedge m = 1 \wedge \text{true}) \vee \\
(m_{\mathrm{PV}}, w_{\mathrm{PV}}, c, t^{c}, y^{c}, m_{\mathrm{S}} \vDash c > 0 \wedge m = 1 \wedge \text{true})
\end{array}\right\} \therefore \\
\left\{ (m_{\mathrm{PV}} \vDash \text{true}) \star \left(\begin{array}{l}
(w_{\mathrm{PV}}, c, t, m_{\mathrm{S}} \vDash c = 0 \wedge m = 1) \vee \\
(w_{\mathrm{PV}}, c, t^{c}, y^{c}, m_{\mathrm{S}} \vDash c > 0 \wedge m = 1)
\end{array}\right) \right\}
\end{array}\right) \\
\{m_{\mathrm{PV}} \vDash \text{true}\}
\end{array}
$$

Fig. 6. Proof of the reader epilogue

The invariant of $w$ is very straightforward:

$$(w_S \vDash w = 0) \vee (w_S, y \vDash w = 1) \tag{32}$$

The invariant of $m$ is a little more intricate:

$$\begin{gathered}(m_S \vDash m = 0) \vee (m_S, w_{PV}, c, t \vDash m = 1 \wedge c = 0) \vee \\ (m_S, w_{PV}, c, t^c, y^c \vDash m = 1 \wedge c > 0)\end{gathered} \tag{33}$$

We need the ticket $t$ because the variable $y$, which is standing for the critical resource controlled by $m$ and $w$, isn't always in the $m$ semaphore. If the reader has a $t^>$ permission then we can be sure that the $m$ semaphore isn't in the $m_S, w_{PV}, c, t \vDash m = 1 \wedge c = 0$ state, because that would claim too much $t$ resource. That is, the $t^>$ ticket guarantees that $c > 0$ when we enter the epilogue.

The really interesting thing about readers-and-writers is that the writer's critical section is mutually exclusive with the "reading happens" section, which isn't protected by a mutex. Of course mutual exclusion is provided by the separation property: a reader's possession of $y^>$ guarantees that no writer can have $y^0$, and vice-versa for a writer. Figure 5 shows how the readers prologue proof goes.

That proof is, once again, very tedious. It could be reduced a little in length if I'd made use of the frame rule in one or two places, but this kind of careful accounting is just what a compiler might be good at, and we can reasonably hope that at least resource safety could be checked by a compiler.

The epilogue proof, shown in figure 6, is similarly tedious. The ticket permission $t^>$ plays the same rôle as it did in the simpler case of figure 3.

This proof is complete in every detail, apart from the issue of how readers share the $m_{PV}$ permission. It even supports the daft things that I claimed you could do in [4]. For example you can write

$$\mathit{pro}; \mathit{pro}; \mathit{pro}; \left(\mathit{read}_1; \mathit{epi} \,\middle\|\, \mathit{read}_2 \,\middle\|\, \mathit{read}_3\right); \mathit{epi}; \mathit{read}_4; \mathit{epi} \tag{34}$$

– and the tickets perform correctly. You can even run two epilogues in parallel (I'd need to account for division of $_P$ and $_V$ permissions to prove it, but that's an unimportant tweak):

$$\mathit{pro}; \mathit{pro}; \mathit{pro}; \left(\mathit{read}_1; \mathit{epi} \,\middle\|\, \mathit{read}_2 \,\middle\|\, \mathit{read}_3; \mathit{epi}\right); \mathit{read}_4; \mathit{epi} \tag{35}$$

## 10 Readers and writers with priority

The program in figure 4 has a well-known deficiency: it doesn't operate any policy to manage readers and writers together. Readers might be starved by a procession of writers, or writers by readers. We might want to give priority to

```
READERS                                  WRITER

P(m3);
  P(r);
    P(m1);
      nr++;                              P(m2);
      if nr = 1 then P(w) else skip fi;    nw++;
    V(m1);                                 if nw = 1 then P(r) else skip fi;
  V(r);                                  V(m2);
V(m3);                                   P(w);

... reading happens here ...             ... writing happens here ...

P(m1);                                   V(w);
  nr−−;                                  P(m2);
  if nr = 0 then V(w) else skip fi         nw−−;
V(m1)                                      if nw = 0 then V(r) else skip fi
                                         V(m2)
```

Fig. 7. Readers and writers with priority to writers (from [9], with shortened names)

```
READERS                                        WRITER

P(e);                                          P(e);
if nw ≠ 0 ∨ dw ≠ 0 then (dr++; V(e);           if nr ≠ 0 ∨ nw ≠ 0 then (dw++; V(e);
                         P(r); dr−−)                                    P(w); dw−−)
                    else skip fi;                                  else skip fi;
nr++;                                          nw := 1;
if dr ≠ 0 ∧ dw = 0 then V(r) else V(e) fi;     V(e);

... reading happens here ...                   ... writing happens here ...

P(e);                                          P(e);
nr−−;                                          nw := 0;
if nr = 0 ∧ dw ≠ 0 then V(w)                   if dr ≠ 0 then V(r)
 elsf dr ≠ 0 ∧ dw = 0 then V(r) else V(e) fi    elsf dw ≠ 0 then V(w) else V(e) fi
```

Fig. 8. Readers and writers with alternating priority (from [2], with modifications)

writers over readers, or vice-versa, and we might want to vary the conditions of priority.

Figure 7 shows the solution given by Courtois et. al. in [9]. It uses rather a lot of semaphores: three mutexes, an $r$ and a $w$. It counts the number of active readers in $nr$ and delayed writers in $nw$. $m1$ and $w$ play the same rôle that $m$ and $w$ did in figure 4. The authors claim that they provide priority to writers over readers, in that at most one process can be waiting at $r$, but that seems a racy argument to me. The first writer waits at $r$ (writer prologue line 3), but a reader could overtake it (reader prologue line 2). It would be nice

$$
\begin{array}{l}
(e_{\mathrm{S}} \vDash e = 0) \vee \\
(e_{\mathrm{S}}, dr, dw, nr, nw, t \vDash e = 1 \wedge dr \geq 0 \wedge dw \geq 0 \wedge nr = 0 \wedge nw = 1) \vee \\
(e_{\mathrm{S}}, dr, dw, nr, nw, t^{nr}, y^{nr} \vDash e = 1 \wedge dr \geq 0 \wedge dw \geq 0 \wedge nr \geq 0 \wedge nw = 0) \\
\\
(r_{\mathrm{S}} \vDash r = 0) \vee \\
(r_{\mathrm{S}}, dr, dw, nr, nw, t^{nr}, y^{nr} \vDash r = 1 \wedge dr > 0 \wedge dw \geq 0 \wedge nr \geq 0 \wedge nw = 0) \\
\\
(w_{\mathrm{S}} \vDash w = 0) \vee \\
(w_{\mathrm{S}}, dr, dw, nr, nw, t, y \vDash w = 1 \wedge dr \geq 0 \wedge dw > 0 \wedge nr = 0 \wedge nw = 0)
\end{array}
$$

Fig. 9. Split-binary semaphore invariants for readers-and-writers with alternating priority

if $r$ could contain the permission to ping $w$, but that doesn't work: a second writer can overtake the one waiting at $r$ and go on to wait at $w$, so it must already have the permission. But the program doesn't seem very interesting anyway, because its proof is essentially the same as the non-prioritised version.

Andrews, in [2], uses a split binary semaphore $e/r/w$. I've modified his solution to try to alternate priority between readers and writers (a reader shouldn't start if there's already a writer waiting; exiting readers should prefer to start waiting writers, and vice-versa) and my version is shown in figure 8. This solution does appear to provide queue management: delayed readers wait on $r$, delayed writers on $w$, both sides can see both counts, and can act 'fairly'. My version puts the instructions that decrement $dr$ and $dw$ in what seems to be the right place, and attempts to alternate readers and writers (if there are writers waiting, the last reader starts one; vice-versa for the writer).

To make a split binary semaphore work in separation logic, as O'Hearn points out in [15], we must make each hold a shared resource when it is 1: then no pair can simultaneously be 1. The invariants of the semaphores are shown in figure 9: $e$ allows for the possibility that $y$ is in the $w$ semaphore; $r$ doesn't have to do that, and has $dr > 0 \wedge nw = 0$; $w$ is similar to $r$ but with $dw > 0 \wedge nr = nw = 0$.

The proofs are pretty straightforward, and largely consist of heaving the invariants from one semaphore to the other. All that I can prove, of course, is safety and mutual exclusion: liveness, and in particular absence of deadlock, are not addressed. Figure 10 shows the reader prologue. The semaphore steps are where all the action is, and deserve attention. P($e$) and P($r$) are straightforward and just like earlier examples. The first V($e$) step requires a little step of consequence: $dr > 0 \rightarrow dr \geq 0$, surely. (That shows that my 'safety' proof doesn't demonstrate very much: it would go through if there were no $dr$++, which is rather important to the correctness of the algorithm.) V($r$) also needs consequence ($nr > 0 \rightarrow nr \geq 0$), as does the second V($e$) to show that the $dr = 0 \vee dw \neq 0$ condition doesn't matter. Figure 11 shows the reader epilogue, which is simpler. The proofs of writer prologue and epilogue are in figures 12 and 13: they are simpler than but ballsachingly similar to the reader proofs.

```
$\{e_{PV}, r_{PV}, w_{PV} \vDash \text{true}\}$
            $\{e_P \vDash \text{true}\}$
Framed: $\left\langle \begin{array}{l} \text{P}(e) \\ \left\{\begin{array}{l}(e_P, dr, dw, nr, nw, t \vDash dr \geq 0 \wedge dw \geq 0 \wedge nr = 0 \wedge nw = 1) \vee \\ (e_P, dr, dw, nr, nw, t^{nr}, y^{nr} \vDash dr \geq 0 \wedge dw \geq 0 \wedge nr \geq 0 \wedge nw = 0)\end{array}\right\}\end{array} \right\rangle$
$\left\{\begin{array}{l}(e_{PV}, r_{PV}, w_{PV}, dr, dw, nr, nw, t \vDash dr \geq 0 \wedge dw \geq 0 \wedge nr = 0 \wedge nw = 1) \vee \\ (e_{PV}, r_{PV}, w_{PV}, dr, dw, nr, nw, t^{nr}, y^{nr} \vDash dr \geq 0 \wedge dw \geq 0 \wedge nr \geq 0 \wedge nw = 0)\end{array}\right\}$
  if $nw \neq 0 \vee dw \neq 0$ then
    $\left\{\begin{array}{l}(e_{PV}, r_{PV}, w_{PV}, dr, dw, nr, nw, t \vDash dr \geq 0 \wedge dw \geq 0 \wedge nr = 0 \wedge nw = 1) \vee \\ (e_{PV}, r_{PV}, w_{PV}, dr, dw, nr, nw, t^{nr}, y^{nr} \vDash dr \geq 0 \wedge dw > 0 \wedge nr \geq 0 \wedge nw = 0)\end{array}\right\}$
      $dr$++
    $\left\{\begin{array}{l}(e_{PV}, r_{PV}, w_{PV}, dr, dw, nr, nw, t \vDash dr > 0 \wedge dw \geq 0 \wedge nr = 0 \wedge nw = 1) \vee \\ (e_{PV}, r_{PV}, w_{PV}, dr, dw, nr, nw, t^{nr}, y^{nr} \vDash dr > 0 \wedge dw > 0 \wedge nr \geq 0 \wedge nw = 0)\end{array}\right\}$
    Framed: $\left\langle \begin{array}{l} \left\{\begin{array}{l}(e_V, dr, dw, nr, nw, t \vDash dr > 0 \wedge dw \geq 0 \wedge nr = 0 \wedge nw = 1) \vee \\ (e_V, dr, dw, nr, nw, t^{nr}, y^{nr} \vDash dr > 0 \wedge dw > 0 \wedge nr \geq 0 \wedge nw = 0)\end{array}\right\} \\ \text{V}(e) \end{array} \right\rangle$
            $\{e_V \vDash \text{true}\}$
    $\{e_{PV}, r_{PV}, w_{PV} \vDash \text{true}\}$
    Framed: $\left\langle \begin{array}{l} \{r_P \vDash \text{true}\} \\ \text{P}(r) \\ \{r_P, dr, dw, nr, nw, t^{nr}, y^{nr} \vDash dr > 0 \wedge dw \geq 0 \wedge nr \geq 0 \wedge nw = 0\} \end{array} \right\rangle$
    $\{e_{PV}, r_{PV}, w_{PV}, dr, dw, nr, nw, t^{nr}, y^{nr} \vDash dr > 0 \wedge dw \geq 0 \wedge nr \geq 0 \wedge nw = 0\}$
      $dr$−−
    $\{e_{PV}, r_{PV}, w_{PV}, dr, dw, nr, nw, t^{nr}, y^{nr} \vDash dr \geq 0 \wedge dw \geq 0 \wedge nr \geq 0 \wedge nw = 0\}$
  else
    $\{e_{PV}, r_{PV}, w_{PV}, dr, dw, nr, nw, t^{nr}, y^{nr} \vDash dr \geq 0 \wedge dw = 0 \wedge nr \geq 0 \wedge nw = 0\}$
      skip
    $\{e_{PV}, r_{PV}, w_{PV}, dr, dw, nr, nw, t^{nr}, y^{nr} \vDash dr \geq 0 \wedge dw = 0 \wedge nr \geq 0 \wedge nw = 0\}$
  fi
$\{e_{PV}, r_{PV}, w_{PV}, dr, dw, nr, nw, t^{nr}, y^{nr} \vDash dr \geq 0 \wedge dw \geq 0 \wedge nr \geq 0 \wedge nw = 0\}$
  $nr$++
$\{e_{PV}, r_{PV}, w_{PV}, dr, dw, nr, nw, t^{nr-1}, y^{nr-1} \vDash dr \geq 0 \wedge dw \geq 0 \wedge nr \geq 0 \wedge nw = 0\}\ \therefore$
$\{e_{PV}, r_{PV}, w_{PV}, dr, dw, nr, nw, t^{nr}, t^{>}, y^{nr}, y^{>} \vDash dr \geq 0 \wedge dw \geq 0 \wedge nr \geq 0 \wedge nw = 0\}$
  if $dr \neq 0 \wedge dw = 0$ then
    $\{e_{PV}, r_{PV}, w_{PV}, dr, dw, nr, nw, t^{nr}, t^{>}, y^{nr}, y^{>} \vDash dr > 0 \wedge dw = 0 \wedge nr \geq 0 \wedge nw = 0\}$
    Framed: $\left\langle \begin{array}{l} \{r_V, dr, dw, nr, nw, t^{nr}, y^{nr} \vDash dr > 0 \wedge dw = 0 \wedge nr \geq 0 \wedge nw = 0\} \\ \text{V}(r) \\ \{r_V \vDash \text{true}\} \end{array} \right\rangle$
    $\{e_{PV}, r_{PV}, w_{PV}, t^{>}, y^{>} \vDash \text{true}\}$
  else
    $\left\{\begin{array}{l} e_{PV}, r_{PV}, w_{PV}, dr, dw, nr, nw, t^{nr}, t^{>}, y^{nr}, y^{>} \vDash \\ \quad dr \geq 0 \wedge dw \geq 0 \wedge nr \geq 0 \wedge nw = 0 \wedge (dr = 0 \vee dw \neq 0)\end{array}\right\}$
    Framed: $\left\langle \begin{array}{l} \{e_V, dr, dw, nr, nw, t^{nr}, y^{nr} \vDash dr \geq 0 \wedge dw \geq 0 \wedge nr \geq 0 \wedge nw = 0 \wedge (dr = 0 \vee dw \neq 0)\} \\ \text{V}(e) \\ \{e_V \vDash \text{true}\} \end{array} \right\rangle$
    $\{e_{PV}, r_{PV}, w_{PV}, t^{>}, y^{>} \vDash \text{true}\}$
  fi
$\{e_{PV}, r_{PV}, w_{PV}, t^{>}, y^{>} \vDash \text{true}\}$
```

Fig. 10. Proof of alternating-priority readers prologue

```
$\{e_{PV}, r_{PV}, w_{PV}, t^{>}, y^{>} \vDash \mathrm{true}\}$
          $\{e_P \vDash \mathrm{true}\}$
Framed: $\left\langle \begin{array}{l} \mathrm{P}(e) \\ \left\{\begin{array}{l}(e_P, dr, dw, nr, nw, t \vDash dr \geq 0 \wedge dw \geq 0 \wedge nr = 0 \wedge nw = 1) \vee \\ (e_P, dr, dw, nr, nw, t^{nr}, y^{nr} \vDash dr \geq 0 \wedge dw \geq 0 \wedge nr \geq 0 \wedge nw = 0)\end{array}\right\} \end{array} \right\rangle$
$\left\{\begin{array}{l}(e_{PV}, r_{PV}, w_{PV}, t^{>}, y^{>}, dr, dw, nr, nw, t \vDash dr \geq 0 \wedge dw \geq 0 \wedge nr = 0 \wedge nw = 1) \vee \\ (e_{PV}, r_{PV}, w_{PV}, t^{>}, y^{>}, dr, dw, nr, nw, t^{nr}, y^{nr} \vDash dr \geq 0 \wedge dw \geq 0 \wedge nr \geq 0 \wedge nw = 0)\end{array}\right\} \therefore$
$\{e_{PV}, r_{PV}, w_{PV}, dr, dw, nr, nw, t^{nr-1}, y^{nr-1} \vDash dr \geq 0 \wedge dw \geq 0 \wedge nr \geq 0 \wedge nw = 0\}$
  $nr--$
$\{e_{PV}, r_{PV}, w_{PV}, dr, dw, nr, nw, t^{nr}, y^{nr} \vDash dr \geq 0 \wedge dw \geq 0 \wedge nr \geq 0 \wedge nw = 0\}$
  if $nr = 0 \wedge dw \neq 0$ then
    $\{e_{PV}, r_{PV}, w_{PV}, dr, dw, nr, nw, t^{nr}, y^{nr} \vDash dr \geq 0 \wedge dw > 0 \wedge nr = 0 \wedge nw = 0\}$
    Framed: $\left\langle \begin{array}{l} \{w_V, dr, dw, nr, nw, t^{nr}, y^{nr} \vDash dr \geq 0 \wedge dw > 0 \wedge nr = 0 \wedge nw = 0\} \\ \quad \mathrm{V}(w) \\ \{w_V \vDash \mathrm{true}\} \end{array} \right\rangle$
    $\{e_{PV}, r_{PV}, w_{PV} \vDash \mathrm{true}\}$
  elsf $dr \neq 0 \wedge dw = 0$ then
    $\{e_{PV}, r_{PV}, w_{PV}, dr, dw, nr, nw, t^{nr}, y^{nr} \vDash dr > 0 \wedge dw = 0 \wedge nr \geq 0 \wedge nw = 0\}$
    Framed: $\left\langle \begin{array}{l} \{r_V, dr, dw, nr, nw, t^{nr}, y^{nr} \vDash dr > 0 \wedge dw = 0 \wedge nr \geq 0 \wedge nw = 0\} \\ \quad \mathrm{V}(r) \\ \{r_V \vDash \mathrm{true}\} \end{array} \right\rangle$
    $\{e_{PV}, r_{PV}, w_{PV} \vDash \mathrm{true}\}$
  else
    $\{e_{PV}, r_{PV}, w_{PV}, dr, dw, nr, nw, t^{nr}, y^{nr} \vDash dr \geq 0 \wedge dw \geq 0 \wedge nr \geq 0 \wedge nw = 0\}$
    Framed: $\left\langle \begin{array}{l} \{e_V, dr, dw, nr, nw, t^{nr}, y^{nr} \vDash dr \geq 0 \wedge dw \geq 0 \wedge nr \geq 0 \wedge nw = 0\} \\ \quad \mathrm{V}(e) \\ \{e_V \vDash \mathrm{true}\} \end{array} \right\rangle$
    $\{e_{PV}, r_{PV}, w_{PV} \vDash \mathrm{true}\}$
  fi
$\{e_{PV}, r_{PV}, w_{PV} \vDash \mathrm{true}\}$
```

Fig. 11. Proof of alternating-priority readers epilogue

## 11 The unbounded buffer: heap separation too

So far I've been discussing a logic that treats variables as resource but has nothing to say about the heap. With a minor tweak it can deal with the heap as well. First, recognise that we don't need stack and heap versions of $\star$: $\mathrm{Own}(x)$ describes a stack-particle and $x \mapsto \_$ describes a heap-particle; $\mathrm{Own}(x) \wedge x \mapsto \_$ doesn't make sense because for sure they aren't the same resource; $\mathrm{Own}(x) \star x \mapsto \_$ it has to be. In the previous discussion, for ease of explanation, I desugared $x1, x2, \ldots, xn \vDash Q$ as $(\mathrm{Own}(x1) \star \mathrm{Own}(x2) \star \cdots \star \mathrm{Own}(xn)) \wedge Q$; if I alter that to $\mathrm{Own}(x1) \star \mathrm{Own}(x2) \star \cdots \star \mathrm{Own}(xn) \star Q$ then I can deal with heap separation too.

The model deals with either version because the stack map $s$ isn't split by $\star$: that is, there isn't a semantic difference between $(\mathrm{Own}(x) \star \mathrm{Own}(y)) \wedge (x > 3 \wedge y < 3)$ and $(\mathrm{Own}(x) \star \mathrm{Own}(y)) \star (x > 3 \wedge y < 3)$ because 'pure' expressions

```
$\{e_{\mathrm{PV}}, r_{\mathrm{PV}}, w_{\mathrm{PV}} \vDash \mathrm{true}\}$
          $\{e_{\mathrm{P}} \vDash \mathrm{true}\}$
Framed: $\Big\langle$ P$(e)$
          $\left\{\begin{array}{l}(e_{\mathrm{P}}, dr, dw, nr, nw, t \vDash dr \geq 0 \wedge dw \geq 0 \wedge nr = 0 \wedge nw = 1) \vee \\ (e_{\mathrm{P}}, dr, dw, nr, nw, t^{nr}, y^{nr} \vDash dr \geq 0 \wedge dw \geq 0 \wedge nr \geq 0 \wedge nw = 0)\end{array}\right\}$ $\Big\rangle$
$\left\{\begin{array}{l}(e_{\mathrm{PV}}, r_{\mathrm{PV}}, w_{\mathrm{PV}}, dr, dw, nr, nw, t \vDash dr \geq 0 \wedge dw \geq 0 \wedge nr = 0 \wedge nw = 1) \vee \\ (e_{\mathrm{PV}}, r_{\mathrm{PV}}, w_{\mathrm{PV}}, dr, dw, nr, nw, t^{nr}, y^{nr} \vDash dr \geq 0 \wedge dw \geq 0 \wedge nr \geq 0 \wedge nw = 0)\end{array}\right\}$
  if $nr \neq 0 \vee nw \neq 0$ then
    $\left\{\begin{array}{l}(e_{\mathrm{PV}}, r_{\mathrm{PV}}, w_{\mathrm{PV}}, dr, dw, nr, nw, t \vDash dr \geq 0 \wedge dw \geq 0 \wedge nr = 0 \wedge nw = 1) \vee \\ (e_{\mathrm{PV}}, r_{\mathrm{PV}}, w_{\mathrm{PV}}, dr, dw, nr, nw, t^{nr}, y^{nr} \vDash dr \geq 0 \wedge dw \geq 0 \wedge nr \geq 0 \wedge nw = 0)\end{array}\right\}$
      $dw$++
    $\left\{\begin{array}{l}(e_{\mathrm{PV}}, r_{\mathrm{PV}}, w_{\mathrm{PV}}, dr, dw, nr, nw, t \vDash dr \geq 0 \wedge dw > 0 \wedge nr = 0 \wedge nw = 1) \vee \\ (e_{\mathrm{PV}}, r_{\mathrm{PV}}, w_{\mathrm{PV}}, dr, dw, nr, nw, t^{nr}, y^{nr} \vDash dr \geq 0 \wedge dw > 0 \wedge nr \geq 0 \wedge nw = 0)\end{array}\right\}$
              $\left\{\begin{array}{l}(e_{\mathrm{V}}, dr, dw, nr, nw, t \vDash dr \geq 0 \wedge dw > 0 \wedge nr = 0 \wedge nw = 1) \vee \\ (e_{\mathrm{V}}, r_{\mathrm{PV}}, w_{\mathrm{PV}}, dr, dw, nr, nw, t^{nr}, y^{nr} \vDash dr \geq 0 \wedge dw > 0 \wedge nr \geq 0 \wedge nw = 0)\end{array}\right\}$
    Framed: $\Big\langle$ V$(e)$ $\Big\rangle$
              $\{e_{\mathrm{V}} \vDash \mathrm{true}\}$
    $\{e_{\mathrm{PV}}, r_{\mathrm{PV}}, w_{\mathrm{PV}} \vDash \mathrm{true}\}$
              $\{w_{\mathrm{P}} \vDash \mathbf{emp}\}$
    Framed: $\Big\langle$ P$(w)$
              $\{w_{\mathrm{P}}, dr, dw, nr, nw, t, y \vDash dr \geq 0 \wedge dw > 0 \wedge nr = 0 \wedge nw = 0\}$ $\Big\rangle$
    $\{e_{\mathrm{PV}}, r_{\mathrm{PV}}, w_{\mathrm{PV}}, dr, dw, nr, nw, t, y \vDash dr \geq 0 \wedge dw > 0 \wedge nr = 0 \wedge nw = 0\}$
      $dw$−−
    $\{e_{\mathrm{PV}}, r_{\mathrm{PV}}, w_{\mathrm{PV}}, dr, dw, nr, nw, t, y \vDash dr \geq 0 \wedge dw \geq 0 \wedge nr = 0 \wedge nw = 0\}$
  else
    $\{e_{\mathrm{PV}}, r_{\mathrm{PV}}, w_{\mathrm{PV}}, dr, dw, nr, nw, t^{nr}, y^{nr} \vDash dr \geq 0 \wedge dw \geq 0 \wedge nr = 0 \wedge nw = 0\}$
      skip
    $\{e_{\mathrm{PV}}, r_{\mathrm{PV}}, w_{\mathrm{PV}}, dr, dw, nr, nw, t, y \vDash dr \geq 0 \wedge dw \geq 0 \wedge nr = 0 \wedge nw = 0\}$
  fi
$\{e_{\mathrm{PV}}, r_{\mathrm{PV}}, w_{\mathrm{PV}}, dr, dw, nr, nw, t, y \vDash dr \geq 0 \wedge dw \geq 0 \wedge nr = 0 \wedge nw = 0\}$
  $nw := 1$
$\{e_{\mathrm{PV}}, r_{\mathrm{PV}}, w_{\mathrm{PV}}, dr, dw, nr, nw, t, y \vDash dr \geq 0 \wedge dw \geq 0 \wedge nr = 0 \wedge nw = 1\}$
          $\{e_{\mathrm{V}}, dr, dw, nr, nw, t \vDash dr \geq 0 \wedge dw \geq 0 \wedge nr = 0 \wedge nw = 1\}$
Framed: $\Big\langle$ V$(e)$ $\Big\rangle$
          $\{e_{\mathrm{V}} \vDash \mathrm{true}\}$
$\{e_{\mathrm{PV}}, r_{\mathrm{PV}}, w_{\mathrm{PV}}, y \vDash \mathrm{true}\}$
```

Fig. 12. Proof of alternating-priority writer prologue

don't feel resource separation. That being so, I can tackle the example which actually started me down the road of considering variables as resource. It doesn't involve variable-ownership transfer, but it does need heap-ownership transfer.

Dijkstra proposed an inter-thread buffer algorithm [10] but thought it would be necessary to make buffer manipulation a critical section. Habermann [11] proved that you don't need to protect buffer manipulation in the

```
$\{e_{\mathrm{PV}}, r_{\mathrm{PV}}, w_{\mathrm{PV}}, y \vDash \mathrm{true}\}$
             $\{e_{\mathrm{P}} \vDash \mathbf{emp}\}$
Framed: $\Bigg\langle$ $\mathrm{P}(e)$
             $\left\{\begin{array}{l}(e_{\mathrm{P}}, dr, dw, nr, nw, t \vDash dr \geq 0 \wedge dw \geq 0 \wedge nr = 0 \wedge nw = 1) \vee \\ (e_{\mathrm{P}}, dr, dw, nr, nw, t^{nr}, y^{nr} \vDash dr \geq 0 \wedge dw \geq 0 \wedge nr \geq 0 \wedge nw = 0)\end{array}\right\}$ $\Bigg\rangle$
$\left\{\begin{array}{l}(e_{\mathrm{PV}}, r_{\mathrm{PV}}, w_{\mathrm{PV}}, y, dr, dw, nr, nw, t \vDash dr \geq 0 \wedge dw \geq 0 \wedge nr = 0 \wedge nw = 1) \vee \\ (e_{\mathrm{PV}}, r_{\mathrm{PV}}, w_{\mathrm{PV}}, y, dr, dw, nr, nw, t^{nr}, y^{nr} \vDash dr \geq 0 \wedge dw \geq 0 \wedge nr \geq 0 \wedge nw = 0)\end{array}\right\}$ $\therefore$
$\{e_{\mathrm{PV}}, r_{\mathrm{PV}}, w_{\mathrm{PV}}, dr, dw, nr, nw, t, y \vDash dr \geq 0 \wedge dw \geq 0 \wedge nr = 0 \wedge nw = 1\}$
  $nw := 0$
$\{e_{\mathrm{PV}}, r_{\mathrm{PV}}, w_{\mathrm{PV}}, dr, dw, nr, nw, t, y \vDash dr \geq 0 \wedge dw \geq 0 \wedge nr = 0 \wedge nw = 0\}$
  if $dr \neq 0$ then
    $\{e_{\mathrm{PV}}, r_{\mathrm{PV}}, w_{\mathrm{PV}}, dr, dw, nr, nw, t, y \vDash dr > 0 \wedge dw \geq 0 \wedge nr = 0 \wedge nw = 0\}$
              $\{dr, dw, nr, nw, t, y \vDash dr > 0 \wedge dw \geq 0 \wedge nr = 0 \wedge nw = 0\}$
    Framed: $\Bigg\langle$   $\mathrm{V}(r)$   $\Bigg\rangle$
              $\{r_{\mathrm{V}} \vDash \mathrm{true}\}$
    $\{e_{\mathrm{PV}}, r_{\mathrm{PV}}, w_{\mathrm{PV}} \vDash \mathrm{true}\}$
  elsf $dw \neq 0$ then
    $\{e_{\mathrm{PV}}, r_{\mathrm{PV}}, w_{\mathrm{PV}}, dr, dw, nr, nw, t, y \vDash dr = 0 \wedge dw > 0 \wedge nr = 0 \wedge nw = 0\}$
              $\{w_{\mathrm{V}}, dr, dw, nr, nw, t, y \vDash dr = 0 \wedge dw > 0 \wedge nr = 0 \wedge nw = 0\}$
    Framed: $\Bigg\langle$   $\mathrm{V}(w)$   $\Bigg\rangle$
              $\{w_{\mathrm{V}} \vDash \mathrm{true}\}$
    $\{e_{\mathrm{PV}}, r_{\mathrm{PV}}, w_{\mathrm{PV}} \vDash \mathrm{true}\}$
  else
    $\{e_{\mathrm{PV}}, r_{\mathrm{PV}}, w_{\mathrm{PV}}, dr, dw, nr, nw, t, y \vDash dr = 0 \wedge dw = 0 \wedge nr = 0 \wedge nw = 0\}$
              $\{e_{\mathrm{V}}, r_{\mathrm{PV}}, w_{\mathrm{PV}}, dr, dw, nr, nw, t, y \vDash dr = 0 \wedge dw = 0 \wedge nr = 0 \wedge nw = 0\}$
    Framed: $\Bigg\langle$   $\mathrm{V}(e)$   $\Bigg\rangle$
              $\{e_{\mathrm{V}} \vDash \mathrm{true}\}$
    $\{e_{\mathrm{PV}}, r_{\mathrm{PV}}, w_{\mathrm{PV}} \vDash \mathrm{true}\}$
  fi
$\{e_{\mathrm{PV}}, r_{\mathrm{PV}}, w_{\mathrm{PV}} \vDash \mathbf{emp}\}$
```

Fig. 13. Proof of alternating-priority writer epilogue

bounded case, using a separation argument (when the buffer count is non-zero, consumer and producer aren't working on the same element). O'Hearn's version [15] of the unbounded buffer algorithm, where cons is a version of new that gives you a two-element cell and $n$ is a counting semaphore holding the buffer, is shown in figure 14, with contextual assertions describing the invariant of consumer, buffer and producer.

The listseg heap predicate describes a sequence of cells in memory with a first and a last pointer. The first pointer points to the first cell in the segment; the last pointer points from the segment to whatever follows (and is not necessarily nil).

$$\begin{array}{rcl} \mathrm{listseg}\ 0\ x\ x & \hat{=} & \mathbf{emp} \\ \mathrm{listseg}\ (n+1)\ x\ y & \hat{=} & \exists x' \cdot (x \mapsto \_, x' \star \mathrm{listseg}\ n\ x'\ y) \end{array} \tag{36}$$

$$\left(\begin{array}{l||l||l}
\text{CONSUMER} & \text{BUFFER} & \text{PRODUCER} \\
\left\{\begin{array}{l} tc, \mathit{front}, f_{0.5}, n_{\mathrm{P}} \\ \quad \vDash \mathit{front} = f \wedge \mathbf{emp} \end{array}\right\} & \{f_{0.5}, b_{0.5}, n_{\mathrm{S}} \vDash \text{listseg } n\ f\ b\} & \left\{\begin{array}{l} tp, \mathit{back}, b_{0.5}, n_{\mathrm{V}} \\ \quad \vDash \mathit{back} = b \wedge \mathit{back} \mapsto \_, \_ \end{array}\right\} \\
\text{while true do} & \textbf{semaphore } n := 0 & \text{while true do} \\
\quad tc := \mathit{front}; & & \quad \mathit{back}.0 := \text{produce}(); \\
\quad \mathrm{P}(n); & \mathrm{P} : \text{with } n \text{ when } n > 0 & \quad tp := \text{cons}(); \\
\quad \mathit{front} := \mathit{front}.1; & \qquad \text{do } n{-}{-};\ f := f.1 \text{ od} & \quad \mathit{back}.1 := tp; \\
\quad \text{consume } tc.0; & \mathrm{V} : \text{with } n \text{ when true} & \quad \mathrm{V}(n); \\
\quad \text{dispose } tc & \qquad \text{do } n{+}{+};\ b := b.1 \text{ od} & \quad \mathit{back} := tp \\
\text{od} & & \text{od}
\end{array}\right)$$

Fig. 14. O'Hearn's unbounded buffer

This predicate doesn't notice the value of the sequence stored in the list; that suits me because I'm only interested in this proof in counting the resource.

Auxiliary variables $f$ and $b$ are beginning and end pointers of the $n$-cell listseg in the buffer. $\mathrm{P}(n)$ blocks as long as $n = 0$; when $n > 0$ it decrements $n$, but it must then give up an element of the buffer list, so it also moves $f$ to preserve the invariant. (We don't need a special implementation of the semaphore to provide mutual exclusion: $f$ is auxiliary, so it can be assigned in no time at all.) $\mathrm{V}(n)$ unguardedly increments $n$ and moves auxiliary $b$.

It's then extremely easy to prove that all three invariants are maintained. Maintenance of the invariant in the body of the consumer loop, for example, is shown in figure 15. I've used forward reasoning in the assignment steps. In the $\mathrm{P}(n)$ step I've shown the whole expanded proof of the CCR step.

This example highlights the fact that the CCR rule (17) needs no side conditions. The side condition of (3), that no other process alters a free variable of antecedent pre- and post-conditions, has become a matter of permissions. The consumer and buffer share a variable $f$; the buffer and producer share another variable $b$; the semaphore operations combine these permissions and allow the semaphore body to alter the relevant variable. Subtle and global has become simple and local.

The unbounded buffer example is also interesting because it has no critical sections. Brinch Hansen certainly thought the possibility of such programs to be a drawback of using semaphores in general, and cites it [6] as one reason for turning to monitors, where there are always visible critical sections. It's not the only reason for making that change — monitors also have the advantage that they allow the programmer to control queuing policies — but it's clear from this example that we now have a means of reasoning formally about 'daring' concurrent programs [15] which don't have critical sections. The need to possess resources taken from a semaphore, and the logical impossibility of duplication of those resources, is sufficient to explain how independent use of shared resource is enforced outside the hardware-established mutual exclusion of the P and V instructions.

```
$\{\mathit{front}, tc, f_{0.5}, n_{\mathrm{P}} \vDash \mathit{front} = f \wedge \mathbf{emp}\}$
  $tc := \mathit{front}$
$\{\mathit{front}, tc, f_{0.5}, n_{\mathrm{P}} \vDash \mathit{front} = f \wedge tc = f \wedge \mathbf{emp}\}$
P$(n)$ : (
    $\left\{ \begin{pmatrix} (\mathit{front}, tc, f_{0.5}, n_{\mathrm{P}} \vDash \mathit{front} = f \wedge tc = f \wedge \mathbf{emp}) \star \\ (f_{0.5}, b_{0.5}, n_{\mathrm{S}} \vDash \text{listseg}\ n\ f\ b) \end{pmatrix} \wedge n > 0 \right\}\ \therefore$
    $\left\{ \begin{array}{l} \mathit{front}, tc, f_{0.5}, n_{\mathrm{P}}, n_{\mathrm{S}}, f_{0.5}, b_{0.5} \vDash \\ \quad \big((\mathit{front} = f \wedge tc = f \wedge \mathbf{emp}) \star \text{listseg}\ n\ f\ b\big) \wedge n > 0 \end{array} \right\}\ \therefore$
    $\{\mathit{front}, tc, f, n_{\mathrm{PS}}, b_{0.5} \vDash f \mapsto \_, f' \star \text{listseg}\ (n-1)\ f'\ b \wedge \mathit{front} = f \wedge tc = f \wedge n > 0\}$
      $n := n - 1$
    $\{\mathit{front}, tc, f, n_{\mathrm{PS}}, b_{0.5} \vDash f \mapsto \_, f' \star \text{listseg}\ n\ f'\ b \wedge \mathit{front} = f \wedge tc = f \wedge n \geq 0\}$
      $f := f.1$
    $\{\mathit{front}, tc, f, n_{\mathrm{PS}}, b_{0.5} \vDash \mathit{front} \mapsto \_, f \star \text{listseg}\ n\ f\ b \wedge tc = \mathit{front} \wedge n \geq 0\}\ \therefore$
    $\{(\mathit{front}, tc, f_{0.5}, n_{\mathrm{P}} \vDash tc = \mathit{front} \wedge \mathit{front} \mapsto \_, f) \star (n_{\mathrm{S}}, f_{0.5}, b_{0.5} \vDash \text{listseg}\ n\ f\ b)\}$
)
$\{\mathit{front}, tc, f_{0.5}, n_{\mathrm{P}} \vDash tc = \mathit{front} \wedge \mathit{front} \mapsto \_, f\}$
  $\mathit{front} := \mathit{front}.1$
$\{\mathit{front}, tc, f_{0.5}, n_{\mathrm{P}} \vDash \mathit{front} = f \wedge tc \mapsto \_, f\}$
  consume $tc.0$
$\{\mathit{front}, tc, f_{0.5}, n_{\mathrm{P}} \vDash \mathit{front} = f \wedge tc \mapsto \_, f\}$
  dispose $tc$
$\{\mathit{front}, tc, f_{0.5}, n_{\mathrm{P}} \vDash \mathit{front} = f \wedge \mathbf{emp}\}$
```

Fig. 15. Invariant preserved in consumer thread

## 12 Conclusions

It does seem as if a bull-headed approach to the variable-ownership problem has made a dent in it. Some rather subtle side conditions have been banished, at least, and formal proof of some classic algorithms is now possible. But the proofs are too complicated to be beautiful and they demonstrate only that the resource accounting works. I think that's alright: we want to be able to build tools that do these proofs so perhaps formalism trumps beauty at this stage of the game.

## Acknowledgements

When this paper was submitted to the MFPS XXI conference, under the single authorship of Richard Bornat, it didn't describe a model of the proposed logic. The MFPS program committee weren't sure about it, but the external referee they consulted was supportive, and even went so far as to provide a model which showed that the logic is at least reasonable. That referee was Hongseok Yang, and his model is the one shown in section 3. His model converted a good syntactic idea into a useful logic, showing even that what at first seemed like counter examples ((20) and (23)) were indeed valid assertions, opening up avenues for exploration.

Although officially an anonymous referee (and he didn't tell me: I guessed) Yang's intervention made this paper worthwhile, and the incorporation of his model makes it right to include him as a co-author, even though the paper was and is written in the first person by Richard Bornat.

For very similar reasons Cristiano Calcagno is listed as co-author too. Weeks before I read Yang's review and saw his model, Cristiano tried to explain a very similar model to me. Had I listened more carefully to him, the paper would have been much stronger from the start. The fact that he had already tried to explain a model to me made me more receptive to Yang's intervention when it arrived, and caused its incorporation into this paper.

I am proud to be associated with Calcagno and with Yang, and I'm grateful that they have agreed to attach their names to this paper. They are significantly responsible for its value, but not responsible for any slips that I made in the proofs or anywhere else. Nor are they to blame for the jokes.

Otherwise, this work arises from long discussions and extended friendly argument with my colleagues in the East London Massive. Josh Berdine tried hard to make $\star$-separation deal with variables as resource. I went on so long about the need to distinguish lv remarks (contexts) from rv remarks (assertions) that I was finally required to put my cards on the table. When I appeared to lose my stake on readers-and-writers Josh Berdine, Cristiano Calcagno and Ivana Mijajlovic helped me find the notion of ticket. Matthew Parkinson pushed me towards the notion of a permission as a ticket. Peter O'Hearn was constructively sceptical throughout. Cristiano Calcagno was encouraging about nested contexts, an earlier idea which didn't quite work, and suggested several models, none of which I understood until Yang eventually knocked some sense into me.

## References

[1] *The jape web site*, http://www.jape.org.uk, latest versions of Jape for various platforms.

[2] Andrews, G., "Concurrent Programming: Principles and Practice," Benjamin Cummings, 1991.

[3] Bornat, R., *Proving pointer programs in Hoare logic*, in: R. C. Backhouse and J. N. Oliveira, editors, *Mathematics of Program Construction, 5th International Conference*, LNCS (2000), pp. 102–126.

[4] Bornat, R., 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* (2005), pp. 259–270.

[5] Boyland, J., *Checking interference with fractional permissions*, in: R. Cousot, editor, *Static Analysis: 10th International Symposium*, Lecture Notes in Computer Science **2694** (2003), pp. 55–72.

[6] Brinch Hansen, P., editor, “The Origin of Concurrent Programming,” Springer-Verlag, 2002.

[7] Brookes, S. D., *A semantics for concurrent separation logic*, in: *CONCUR’04: 15th International Conference on Concurrency Theory*, Lecture Notes in Computer Science **3170** (2004), pp. 16–34, extended version to appear in *Theoretical Computer Science*.

[8] Burstall, R., *Some techniques for proving correctness of programs which alter data structures*, Machine Intelligence **7** (1972), pp. 23–50.

[9] Courtois, P. J., F. Heymans and D. L. Parnas, *Concurrent control with “readers” and “writers”*, Commun. ACM **14** (1971), pp. 667–668.

[10] Dijkstra, E. W., *Cooperating sequential processes*, in: F. Genuys, editor, *Programming Languages*, Academic Press, 1968 pp. 43–112, reprinted in [6].

[11] Habermann, A. N., *Synchronization of communicating processes*, in: *Proceedings of the third ACM symposium on Operating systems principles* (1971), p. 80.

[12] Hoare, C. A. R., *Towards a theory of parallel programming*, in: Hoare and Perrott, editors, *, in , ed. Operating System Techniques,, 1972.*, Academic Press, 1972 pp. 61–71, reprinted in [6].

[13] O’Hearn, P., *Notes on separation logic for shared-variable concurrency* (2002), unpublished.

[14] O’Hearn, P., J. Reynolds and H. Yang, *Local reasoning about programs that alter data structures*, in: L. Fribourg, editor, *CSL 2001* (2001), pp. 1–19, LNCS 2142.

[15] O’Hearn, P. W., *Resources, concurrency and local reasoning*, to appear in *Theoretical Computer Science*; preliminary version published as [16].

[16] O’Hearn, P. W., *Resources, concurrency and local reasoning*, in: *CONCUR’04: 15th International Conference on Concurrency Theory*, Lecture Notes in Computer Science **3170** (2004), pp. 49–67, extended version is [15].

[17] O’Hearn, P. W. and D. J. Pym, *The logic of bunched implications*, Bulletin of Symbolic Logic **5** (1999), pp. 215–244.

[18] Owicki, S. and D. Gries, *Verifying properties of parallel programs: An axiomatic approach*, Comm. ACM **19** (1976), pp. 279–285.

[19] Pym, D., “The Semantics and Proof Theory of the Logic of Bunched Implications,” Applied Logic Series **26**, Kluwer Academic Publishers, 2002.

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

[21] Reynolds, J. C., *Separation logic: A logic for shared mutable data structures*, in: *LICS ’02: Proceedings of the 17th Annual IEEE Symposium on Logic in Computer Science* (2002), pp. 55–74.

[22] Yang, H. and P. O'Hearn, *A semantic basis for local reasoning*, in: *5th FOSSACS* (2002), pp. 402–416.