An admissible second order frame rule in region logic (extended abstract)

David A. Naumann
Stevens Institute of Technology, Hoboken, NJ, USA

March 14, 2008

Abstract

Shared mutable objects and reentrant callacks can subvert encapsulation in object-based programs. For modular reasoning, verifiers rely on methodologies. These combine special annotations or types with instrumentation (ghost state) and syntactic restrictions on programs and specifications, which pose challenges for proving soundness and for comparing/combining methodologies. This paper formalizes a second order frame rule, similar to that of separation logic but for a logic with explicit regions. The rule captures proof obligations of invariant methodologies such as ownership for dynamically instantiable abstractions, using an extended provability judgement that threads a state-based separation discipline through a proof to points where encapsulation is at risk. Soundness with respect to a standard semantics is proved by an admissibility argument.

1 Introduction

An essential technique for reasoning about modular code was articulated by Hoare [8]: Clients of a module are verified using a public specification, say $\{\varphi\}\ m\ \{\varphi'\}$ where m is a method (procedure) name, but the verification condition for the method's body, B, is $\{\varphi\wedge\psi\}\,B\,\{\varphi'\wedge\psi\}$. Here ψ is the module invariant, supposed not to be susceptible to client interference. One way to justify Hoare's mismatch draws on the rule of Invariance, which in its original form says that $\{\varphi\}\,C\,\{\varphi'\}$ implies $\{\varphi\wedge\psi\}\,C\,\{\varphi'\wedge\psi\}$ provided that command C writes no variables read in ψ. A module construct could ensure, for reasons of scope, this disjointness of variables. Then any proof of a client program, using interface specification $\{\varphi\}\,m\,\{\varphi'\}$, could be transformed to one using the specification $\{\varphi\wedge\psi\}\,m\,\{\varphi'\wedge\psi\}$ against which B is verified, by adding instances of Invariance to conjoin ψ at appropriate points.

In the presence of various language features this simple story is subverted because it is difficult to specify and verify the absence of interference. In particular, verification systems for object-oriented programs, e.g., ESC/Java and Spec#, try to provide sound forms of Hoare's mismatch by imposing programming disciplines. These combine special annotations or types with instrumentation (ghost state) and syntactic restrictions on programs and specifications. For example, the Universe Type system [14] provides heap encapsulation via a *confinement invariant* that designates certain objects as owned by others and it *bounds the effects* of client programs, disallowing write access to non-owned objects. The mismatch is then sound for invariants that depend only on owned objects.

This paper studies a logic in which such disciplines can be encoded and deployed on a per-module basis. It supports invariants at varying granularity ranging from a single instance (and its owned representation) to clusters with shared representation, which occur in various important design patterns [10]. The mismatch enabled by a discipline is made explicit in a *second order frame rule* adapted from that of separation logic [17]. The ordinary frame rule of separation logic resembles the Invariance rule, but without using a side condition for absence of interference: it says that $\{\varphi\}\,C\,\{\varphi'\}$ implies $\{\varphi * \psi\}\,C\,\{\varphi' * \psi\}$ where $\varphi * \psi$ says the truth of φ and ψ is supported by disjoint footprints in the heap. The antecedent $\{\varphi\}\ C\ \{\varphi'\}$ bounds the effects of C; loosely speaking, its footprint is within the confines of φ and φ'. The second order frame rule lifts this reasoning to the level of assumptions, distilling the essence of Hoare's mismatch.

In separation logic, the second order frame rule has nontrivial interaction with the ordinary rule of conjunctivity, owing to the existential implicit in the semantics of $*$. The soundness result in [17] restricts the rule to *precise* invariants ψ, i.e., with uniquely determined footprints. Soundness, without the restriction to precise predicates, has been proved in semantic models that do not validate the rule of conjunctivity [3, 18].

The logic in this paper uses regions to make footprints explicit. Regions have been used in several calculi to track disjointness of heap effects for various purposes [19]. Here, an ordinary first order assertion language is augmented with type **rgn** for finite sets of allocated references, following Kassios [9]. He advocates the use of ghost variables and fields of type **rgn** and a second order predicate, R **frames** ψ, that says the footprint of ψ in the current state is within the region currently denoted by expression R. Working in a relational calculus of refinement [7], Kassios shows by example how regions support explicit formulation of existing and novel disciplines, without additional machinery and without imposing one discipline on all modules.

Aiming to adapt Kassios' insights to conventional verifiers using specification languages like JML, Banerjee et al [1] introduce a Hoare logic where effects in the "modifies clause" can refer to region expressions. Their logic relies on a subsidiary judgement of the form $\varphi \vdash \overline{\varepsilon}$ **frm** ψ that says the read effect or footprint of formula ψ is conservatively approximated by the set of effects $\overline{\varepsilon}$, in any state satisfying φ (where φ, ψ are ordinary first order formulas). Their frame rule has an explicit proviso about disjointness, expressed in terms of the footprint of ψ and the write effect of the command.

In this paper, their logic is extended with a second order frame rule. The rule relies on a judgement form that serves to thread a confinement invariant and client effect bound — which together embody some discipline for encapsulation— through the part of a program proof in which client code is verified against assumed specifications that do not mention the module invariant. Our main result implies the soundness of this logic with respect to a standard denotational semantics. The second order frame rule is treated indirectly: it is proved to be *admissible*, i.e., any proof can be transformed (in the manner sketched above) to one not using it.

The second order frame rule and its relevance to programming methodologies is described informally in Banerjee et al. [1]. But no details are given about the extended judgements and other rules needeed to use the second order frame rule.

Outline. Sec. 2 illustrates the use of regions for reasoning in the small, to introduce the partial correctness logic with regions [1] which is reviewed in Sec. 3–5. It then states and briefly describes the second order frame rule. Sec. 6 formalizes the extended logic. Its soundness is proved in Sec. 7. Sec. 8 has concluding remarks. A longer version of the paper can be found online at `http://www.cs.stevens.edu/~naumann/pub/hofrl2.pdf`.

2 Framing and footprints with explicit regions

Let Γ be the context $x{:}\, K, y{:}\, K, r{:}\, \mathbf{rgn}$ where K is some class with field $f{:}\, K$. Values of type **rgn** are finite sets of allocated references. Expression $\langle y\rangle$ denotes a singleton region or the empty set, depending on whether y is **null**, and $y \in r$ abbreviates $y \neq \mathbf{null} \wedge \langle y\rangle \subseteq r$. The axiom for field update (plus consequence rule to add $y \notin r$) yields

$$\{y \neq \mathbf{null} \wedge y \notin r\}\ y.f{:}= x\ \{y.f = x\}[\mathbf{wr}\langle y\rangle{\bullet}f] \tag{1}$$

The effect $\mathbf{wr}\langle y\rangle{\bullet}f$ says that the command updates no state other than, possibly, the f field of an object (reference) in $\langle y\rangle$. Primitive formula $r{\bullet}f \subseteq r$ says r is closed under dereferences of field f. We will show it is preserved by the update, using the rule

$$\text{FRAME}\quad \frac{\{\varphi\}\ C\ \{\varphi'\}[\overline{\varepsilon}] \qquad \varphi \vdash \overline{\delta}\ \mathbf{frm}\ \psi \qquad \varphi \Rightarrow \overline{\delta} \star \overline{\varepsilon}}{\{\varphi \wedge \psi\}\ C\ \{\varphi' \wedge \psi\}[\overline{\varepsilon}]}$$

The operator $\star$ generates a conjunction of region disjointness formulas sufficient to ensure that the writes on the right cannot interfere with the reads on the left, so $\varphi \Rightarrow \overline{\delta} \star \overline{\varepsilon}$ says that footprint $\overline{\delta}$ of ψ is disjoint from the writes of C. Antecedent φ can be assumed since the regions mentioned in reads and writes are interpreted in the initial state.[1] For the example, $\mathbf{true} \vdash \mathbf{rd}\, r, \mathbf{rd}\, r{\bullet}f\ \ \mathbf{frm}\ \ r{\bullet}f \subseteq r$ and the separator formula, $\mathbf{rd}\, r, \mathbf{rd}\, r{\bullet}f\ \star\ \mathbf{wr}\langle y\rangle{\bullet}f$, is $r \mathbin{\#} \langle y\rangle$. Now $r \mathbin{\#} \langle y\rangle$ just says the two sets are disjoint, which follows from $y \notin r$, so (1) yields $\{y \neq \mathbf{null} \wedge y \notin r \wedge r{\bullet}f \subseteq r\}\ y.f{:}= x\ \{r{\bullet}f \subseteq r\}[\mathbf{wr}\langle y\rangle{\bullet}f]$ by an application of FRAME.

Here is the second order frame rule, for reasoning about commands that may invoke procedures of some module with encapsulated state.

$$\text{FRAME2}\quad \frac{\Delta; \Delta^* \vdash^{\Gamma}_{\chi;\overline{\eta}} \{\varphi\}\ C\ \{\varphi'\}[\overline{\varepsilon}; \overline{\varepsilon}^*] \qquad \chi \vdash \overline{\delta}\ \mathbf{frm}\ \psi \qquad \chi \Rightarrow \overline{\delta} \star \overline{\eta}}{\Delta, (\Delta^* \otimes \psi) \vdash^{\Gamma} \{\varphi \wedge \psi\}\ C\ \{\varphi' \wedge \psi\}[\overline{\varepsilon}, \overline{\varepsilon}^*]}$$

In part, the antecedent says some client C is verified with respect to an ambient library, Δ, and some specifications Δ^* treated as a module with hidden invariant ψ. The consequent

[1]A similar effect is achieved in separation logic using the separated conjunctions $\varphi * \psi$ and $\varphi' * \psi$, although it disallows the reads of C to overlap the reads of ψ (this can be addressed using permissions [5]).
Aside: Given frames $\overline{\varepsilon}_\varphi$ for φ and $\overline{\varepsilon}_\psi$ for ψ, we can approximate $\varphi * \psi$ as $\varphi \wedge \psi \wedge (\overline{\varepsilon}_\varphi \star \overline{\delta})$ where $\overline{\delta}$ is obtained from $\overline{\varepsilon}_\psi$ by turning **rd** into **wr**. In a given state, the footprints $\overline{\varepsilon}_\varphi$ and $\overline{\varepsilon}_\psi$ have fixed interpretations, whereas $\varphi * \psi$ allows that there may be two ways of partitioning the heap in support of φ and ψ. Our logic's use of explicit footprints and ghost variables can be seen as skolemizing the existential implicit in $*$.

says that the implementations are subject to proof obligation $\Delta^* \otimes \psi$ which means ψ is added as explicit pre- and post-condition of each specification in Δ^*. Once those assumptions are discharged, what has been verified about C is $\{\varphi \wedge \psi\}\ C\ \{\varphi' \wedge \psi\}[\ldots]$. This is enough, because in a complete program ψ is established by module initialization.

The formula χ is called the *confinement invariant* and $\overline{\eta}$ is the *client effect bound*. Together, these capture some discipline whereby the module invariant ψ is protected from interference by clients. The antecedent correctness judgement of Frame2 is in an extended logic, EL, that propagates χ and $\overline{\eta}$ through the subproof. The EL judgements also separate the effects ($\overline{\varepsilon}$) of "client code", which must be bounded by $\overline{\eta}$, from the effects ($\overline{\varepsilon}^*$) of the module procedures which are likely to write state read by ψ. Condition $\chi \Rightarrow \overline{\delta} \star \overline{\eta}$ says that, in χ-states, effects within the client effect bound $\overline{\eta}$ do not interfere with the frame of ψ, which is given by $\chi \vdash \overline{\delta}\ \mathbf{frm}\ \psi$. These two side conditions in the rule are what is necessary for the discipline embodied by $\chi, \overline{\eta}$ to be sound.

Current verifiers typically impose a single discipline on all programs, so one can imagine that the proof for each method body uses a single instance of Frame2, as its last step. But the logic EL allows multiple uses of Frame2, so that different sub-proofs can involve different disciplines. We omit the straightforward generalization to allow the use of more than one module: the context (resp. effect) would be a map from module identifiers to specs (resp. effect sets).

A type-based [14] or assertion-based [12, 16] discipline will impose the confinement invariant and client effect bound at every program point. This is not necessary: Both can be violated within large subprograms that do not have calls to methods in Δ^*, as our main result shows. Such subprograms are verified within the RL sub-logic of EL, via the rule RLtoEL given in Sec. 6.

In Sect. 6 we formalize the extended judgements used in the antecedent of Frame2. However, our semantics of these judgements does not fully capture the intended meaning that the client respects the discipline embodied by $\chi, \overline{\eta}$ —indeed, rule Frame2 is patently unsound for our "forgetful" semantics. Our main result dodges the semantic question, by showing soundness of the logic EL including rule Frame2.

3 Illustrative programming language

The formalization caters for the intended application to Java-like programming languages and specification languages like JML and Spec#, in that we consider typed, first-order objects. To streamline the logic, however, methods are not bound to classes and only a rudimentary module construct is considered. A simple denotational semantics is used as in [1] (cf. [11]).

Assume we are given a *class table*, i.e., declarations of a number of named record types. Record fields, like method parameters and program variables, are assigned data types: **int**, **rgn**, or class name. Fields of a class may make mutually recursive reference to other classes. We omit concrete syntax for classes; *ClassName* is the set of names of declared classes, fields(K) gives the field declarations for class K. Field names are assumed globally unique.

$$
\begin{array}{lll}
T & ::= \mathbf{int} \mid K \mid \mathbf{rgn} & \text{for } K \in \mathit{ClassName} \\
E & ::= x \mid c \mid \mathbf{null} \mid E{\oplus}E & \text{for } \oplus \in \{=,+,>,\ldots\},\ c \in \mathbb{Z} \\
R & ::= x \mid x.f \mid \langle E\rangle \mid \mathbf{emp} \mid \mathbf{all} \mid R{\otimes}R & \text{for } \otimes \in \{\cup,\cap,-\} \\
F & ::= E \mid R & \\
M & ::= m(\overline{x}{:}\,\overline{T}) & \text{for } m \in \mathit{MethodName} \\
C & ::= m(\overline{z}) \mid \mathbf{let}\ \overline{M}\ \mathbf{be}\ \overline{C}\ \mathbf{in}\ C \mid x{:=}\,F \mid x{:=}\,\mathbf{new}\,K \mid x{:=}\,x.f \mid x.f{:=}\,F & \\
 & \quad \mid \mathbf{if}\ x\ \mathbf{then}\ C\ \mathbf{else}\ C \mid \mathbf{while}\ x\ \mathbf{do}\ C \mid C\,;C \mid \mathbf{var}\ x{:}\,T\ \mathbf{in}\ C\ \mathbf{end} &
\end{array}
$$

Figure 1: Programming language. Let $x, y, r \in \mathit{VarName}$ and $f, g, h \in \mathit{FieldName}$.

A main program is a command C in the context of some variable and method declarations, written $\Gamma; \overline{M} \vdash C$. Here Γ is a finite map from variable names to data types. Finite maps are also written, e.g., $\overline{x}{:}\,\overline{T}$ for a list of variable declarations; comma means union of disjoint maps and binds tightly. The signatures of methods in scope are given by the list $\overline{M}$. Parameters are passed by value and methods do not have a result. For the typing judgement $\Gamma; \overline{M} \vdash C$ to be well formed, all free variables of C must be in $\mathrm{dom}(\Gamma)$ and no method name is duplicated in $\overline{M}$. Programs should be typable as usual for Java-like languages (e.g., [11]), disallowing pointer arithmetic.

N.B. We assume the *hygiene condition* that no variable occurs both bound and free.

The grammar is in Fig. 1. As in separation logic, ordinary expressions E do not depend on the heap; command $x{:=}\,y.f$ is included for reading a field. Region expressions R can have one-step dependence, in the case of $x.g$ where field g has type **rgn**. Effects and formulas involve the form $R{\bullet}f$ but this is *not* a region expression. The only operations on expressions of type **rgn** are $\cup, \cap, -$. These operations all return type **rgn**, so it is clear region variables/fields are only ghosts for reasoning.

The construct **let** $\overline{N}$ **be** $\overline{C}$ **in** C binds a list $\overline{N}$ of method signatures to a same-length list $\overline{C}$ of commands that may make mutually recursive reference to each other, may use global variables, and may be used in the body C.

$$
\begin{array}{lll}
[\![r]\!]\sigma & = & \sigma(r) \\
[\![x.g]\!]\sigma & = & \sigma(x.g) \text{ if } \sigma(x) \neq \text{nil, else } \varnothing \\
[\![\langle E\rangle]\!]\sigma & = & \{\ [\![E]\!]\sigma\ \} \text{ if } [\![E]\!]\sigma \neq \text{nil, else } \varnothing \\
[\![R_1{\cup}R_2]\!]\sigma & = & [\![R_1]\!]\sigma \cup [\![R_2]\!]\sigma \\
[\![R_1{-}R_2]\!]\sigma & = & [\![R_1]\!]\sigma - [\![R_2]\!]\sigma \\
[\![\mathbf{emp}]\!]\sigma & = & \varnothing \\
[\![\mathbf{all}]\!]\sigma & = & \mathrm{alloc}(\sigma)
\end{array}
$$

Figure 2: Semantics, $[\![\Gamma \vdash R{:}\ \mathbf{rgn}]\!]$, for $g{:}\,\mathbf{rgn}$.

A *state*, σ, consists of an assignment of types to the allocated object references, a heap mapping allocated references to object states, and a store assigning values to the variables in scope.[2] A *state transformer for* Γ is a total function that sends Γ-states to the outcome $\bot$ (for divergence or undefinedness), $⫛$ (for runtime errors), or a state. Outcomes are ordered with $⫛$ incomparable to states and $\bot$ below all. We say σ' *extends* σ provided $\mathrm{alloc}(\sigma) \subseteq \mathrm{alloc}(\sigma')$ and

[2]We use the following notations: $\sigma(x)$ is the value of x in σ; $\sigma(x.f)$ is the value of field f of object $\sigma(x)$; $\sigma(o.f)$ is the value of f of reference value o; $\mathrm{alloc}(\sigma)$ is the set of all allocated references; $\mathrm{type}(o, \sigma)$ is the allocated type of reference o; $\mathrm{extend}(\sigma, x, v)$ is the state like σ but with variable x (not present in σ) added with value v.

$$\begin{array}{rcl} \varphi, \psi, \chi & ::= & E = E \mid x.f = F \mid R \subseteq R \mid R \mathbin{\#} R \mid R\bullet f \mathbin{\#} R \mid R\bullet f \subseteq R \\ & \mid & (\forall x\colon \textbf{int} \mid \varphi) \mid (\forall x\colon K \in R \mid \varphi) \mid \varphi \wedge \varphi \mid \neg\varphi \end{array}$$

$$\begin{array}{lll} \sigma \models x.f = F & \text{iff} & \sigma(x) \neq \text{nil and } \sigma(x.f) = [\![F]\!]\sigma \\ \sigma \models R_1 \mathbin{\#} R_2 & \text{iff} & [\![R_1]\!]\sigma \cap [\![R_2]\!]\sigma = \varnothing \\ \sigma \models R_1\bullet f \subseteq R_2 & \text{iff} & \sigma(o.f) = \text{nil or } \sigma(o.f) \in [\![R_2]\!]\sigma \quad \text{for all } o \in [\![R_1]\!]\sigma \text{ with } f \in \mathit{re}\text{fields}(o,\sigma) \\ \sigma \models R_1\bullet f \mathbin{\#} R_2 & \text{iff} & \sigma(o.f) \notin [\![R_2]\!]\sigma \quad \text{for all } o \in [\![R_1]\!]\sigma \text{ with } f \in \mathit{re}\text{fields}(o,\sigma) \\ \sigma \models^{\Gamma} \forall x\colon K{\in}R \mid \varphi & \text{iff} & \text{extend}(\sigma, x, o) \models^{\Gamma, x:K} \varphi \quad \text{for all } o \text{ in } [\![R]\!]\sigma \text{ with type}(o,\sigma) = K \end{array}$$

Figure 3: Formulas and selected semantics. Using *re*fields(o, σ) for the names of reference-type fields in type(o, σ).

type$(o, \sigma) = $ type(o, σ') for all $o \in$ alloc(σ). Commands denote state transformers with the property that the final state, if any, extends the initial one.

A *method environment for* $\Gamma, \overline{M}$ maps each method name m in $\overline{M}$ to a state transformer for $\Gamma, \overline{x}\colon \overline{T}$ where $\overline{x}\colon \overline{T}$ are the parameters of m in $\overline{M}$. The semantics $[\![\Gamma; \overline{M} \vdash C]\!]$ takes a method environment for $\Gamma, \overline{M}$ to a state transformer for Γ, defined by recursion on the structure of C. Details of the semantics are omitted, except for region expressions, see Fig. 2. Null dereference is not an error for region type fields, whereas in the primitive command $x\colon= y.f$ with f not of region type, the program yields $\pitchfork$ in case y is null. The semantics is deterministic. We assume given a choice function for fresh references.

4 Assertions, effects, and separators

The assertion language is first order and does not allow variables of type **rgn** to be bound by quantifiers. The syntax and semantics are given in Fig. 3 (see [1] for typing). It is straightforward to add inductive definitions, as is needed for precise functional specifications. But interesting separation properties can be expressed without them, using the one-step points-to formulas for disjointness ($R\bullet f \mathbin{\#} R'$) and inclusion ($R\bullet f \subseteq R'$). A formula is *valid* if it holds in all states.

An *effect set* is written as a comma-separated list, $\overline{\varepsilon}$, of *effects* given by the grammar

$$\varepsilon ::= \textbf{rd}\, x \mid \textbf{wr}\, x \mid \textbf{rd}\, R\bullet f \mid \textbf{wr}\, R\bullet f \mid \textbf{rd}\,\textbf{all} \mid \textbf{wr}\,\textbf{all} \mid \textbf{fr}\, R$$

We let $\delta, \varepsilon, \eta$ range over effects. Effects must be well formed (wf) for the context Γ in which they occur: $\textbf{rd}\, x$ and $\textbf{wr}\, x$ are wf if $x \in \text{dom}(\Gamma)$; $\textbf{rd}\, R\bullet f$, $\textbf{wr}\, R\bullet f$, and $\textbf{fr}\, R$ are wf if R is wf in Γ. Effect **wr all** allows allocation, since **all** denotes the region of all currently allocated objects. Effect $\textbf{fr}\, R$ says that the final value of R contains only freshly allocated objects.

Besides their use for formulas, read effects in method specifications are useful for various purposes. For lack of space in this extended abstract, we omit read effects from the command correctness judgements and rules.[3]

[3]They are straightforward and not pertinent to our results, with the exception that the substitution rule uses read effects for variables.

$$\frac{\varphi \text{ is primitive}}{\mathbf{true} \vdash \text{ftpt}(\varphi)\ \mathbf{frm}\ \varphi} \qquad \frac{\varphi \vdash \overline{\varepsilon}\ \mathbf{frm}\ \psi \qquad \psi \Leftrightarrow \chi}{\varphi \vdash \overline{\varepsilon}\ \mathbf{frm}\ \chi}$$

$$\frac{\varphi \vdash \overline{\varepsilon}\ \mathbf{frm}\ \chi \qquad \varphi \wedge \chi \vdash \overline{\varepsilon}\ \mathbf{frm}\ \psi}{\varphi \vdash \overline{\varepsilon}\ \mathbf{frm}\ \chi \wedge \psi} \qquad \frac{\varphi \vdash \overline{\varepsilon}\ \mathbf{frm}\ \chi \qquad \varphi \vdash \overline{\varepsilon} \leq \overline{\eta} \qquad \psi \Rightarrow \varphi}{\psi \vdash \overline{\eta}\ \mathbf{frm}\ \chi}$$

Figure 4: Selected rules for frames judgement.

Let effect set $\overline{\varepsilon}$ be well formed in Γ and let σ, σ' be Γ-states. We say $\overline{\varepsilon}$ *allows transition from σ to σ'*, written $\sigma \to \sigma' \models \overline{\varepsilon}$, iff σ' extends σ and the following all hold:

(T0) for every y in $\text{dom}(\Gamma)$, either $\sigma(y) = \sigma'(y)$ or $\mathbf{wr}\, y$ is in $\overline{\varepsilon}$
(T1) for every $o \in \text{alloc}(\sigma)$ and every $f \in \text{fields}(o, \sigma)$, either $\sigma(o.f) = \sigma'(o.f)$ or there is $\mathbf{wr}\, R\bullet f$ in $\overline{\varepsilon}$ such that $o \in [\![R]\!]\sigma$
(T2) if $\text{alloc}(\sigma') \neq \text{alloc}(\sigma)$ then $\mathbf{wr}\,\mathbf{all}$ is in $\overline{\varepsilon}$
(T3) for every $\mathbf{fr}\, R$ in $\overline{\varepsilon}$: $[\![R]\!]\sigma' \subseteq \text{alloc}(\sigma') - \text{alloc}(\sigma)$

Let $\overline{\varepsilon}$ be an effect set and σ, σ' be states such that σ' extends σ. Say that σ and σ' *agree on $\overline{\varepsilon}$* provided the following hold:

(A0) for all $\mathbf{rd}\, x$ in $\overline{\varepsilon}$, we have $\sigma(x) = \sigma'(x)$
(A1) if $\mathbf{rd}\,\mathbf{all}$ in $\overline{\varepsilon}$ then $\text{alloc}(\sigma) = \text{alloc}(\sigma')$
(A2) for all $\mathbf{rd}\, R\bullet f$ in $\overline{\varepsilon}$, for all $o \in [\![R]\!]\sigma$ with $f \in \text{fields}(o, \sigma)$, we have $\sigma(o.f) = \sigma'(o.f)$

The judgement $\varphi \vdash \overline{\varepsilon} \leq \overline{\eta}$ expresses that the writes/reads in $\overline{\eta}$ permit more program behaviors than $\overline{\varepsilon}$ does. It has a straightforward induction definition, with base cases such as $\mathbf{true} \vdash \overline{\varepsilon} \leq \overline{\varepsilon}, \varepsilon$ and $R_1 \subseteq R_2 \vdash \mathbf{wr}\, R_1\bullet f \leq \mathbf{wr}\, R_2\bullet f$.

Framing. For ordinary expressions, define $\text{ftpt}(E) = \{\mathbf{rd}\, x \mid x \in \mathit{Vars}(E)\}$. For the read effects of region expressions and primitive assertions, $\text{ftpt}(R)$ is recursively, e.g.,

$$\begin{array}{ll} \text{ftpt}(x) = \{\mathbf{rd}\, x\} & \text{ftpt}(\langle E \rangle) = \text{ftpt}(E) \\ \text{ftpt}(\mathbf{all}) = \{\mathbf{rd}\,\mathbf{all}\} & \text{ftpt}(R_1 \cap R_2) = \text{ftpt}(R_1) \cup \text{ftpt}(R_2) \\ \text{ftpt}(x.g) = \{\mathbf{rd}\, x, \mathbf{rd} \langle x \rangle \bullet g\} \text{ (where } g\colon \mathbf{rgn}) & \text{ftpt}(R_1\bullet f \subseteq R_2) = \text{ftpt}(R_1) \cup \text{ftpt}(R_2) \cup \{\mathbf{rd}\, R_1\bullet f\} \end{array}$$

Fig. 4 defines the *frames* judgement $\varphi \vdash \overline{\varepsilon}\ \mathbf{frm}\ \varphi'$. The rule for $\forall$ exploits the bounding region to give a footprint that is "local", unless the bound is **all**.

For a primitive, $\text{ftpt}(\varphi)$ is precisely what is needed to evaluate the formula. For any formula, the frames judgement gives a conservative approximation of what must be read.

Given effect sets $\overline{\varepsilon}$ and $\overline{\eta}$, the *separator* formula $\overline{\varepsilon} \star \overline{\eta}$ is a conjunction of certain disjointnesses. In a state where $\overline{\varepsilon} \star \overline{\eta}$ holds, nothing that is allowed by $\overline{\varepsilon}$ to be read can be written according to $\overline{\eta}$. We define $\overline{\varepsilon} \star \overline{\eta}$ by recursion on effect sets: It distributes conjunctively, by $(\varepsilon, \overline{\varepsilon}) \star \overline{\varepsilon}_1 = (\varepsilon \star \overline{\varepsilon}_1) \wedge (\overline{\varepsilon} \star \overline{\varepsilon}_1)$ and $\varepsilon \star (\varepsilon', \overline{\varepsilon}) = (\varepsilon \star \varepsilon') \wedge (\varepsilon \star \overline{\varepsilon})$. For

single effects the definition is:
$$\begin{array}{lcl}\mathbf{rd}\, y \star \mathbf{wr}\, x & = & \text{if } x \equiv y \text{ then } \mathbf{false} \text{ else } \mathbf{true} \\ \mathbf{rd}\, R_1 \bullet f \star \mathbf{wr}\, R_2 \bullet g & = & \text{if } f \equiv g \text{ then } R_1 \mathbin{\#} R_2 \text{ else } \mathbf{true} \\ \mathbf{rd}\, \mathbf{all} \star \mathbf{wr}\, \mathbf{all} & = & \mathbf{false} \\ \varepsilon \star \varepsilon' & = & \mathbf{true} \quad \text{otherwise}\end{array}$$
Note that write effects in $\overline{\varepsilon}$, reads in $\overline{\eta}$, and freshnesses in both, are not relevant.

Soundness of rule FRAME hinges on two semantic properties. First, if $\overline{\delta}$ frames φ and states σ, σ' agree on $\overline{\delta}$ then they agree on the truth value of φ. Second, if transition from σ to σ' is allowed by $\overline{\varepsilon}$ and σ satisfies $\overline{\delta} \star \overline{\varepsilon}$ then σ, σ' agree on $\overline{\delta}$.

5 The core program logic, RL

The RL judgement for correctness takes the form

$$\Delta \vdash^{\Gamma} \{\varphi\}\ C\ \{\varphi'\}[\overline{\varepsilon}] \tag{2}$$

where Δ is a set of *method specifications*, each of the form

$$(\overline{y}\colon \overline{U})\{\psi\}m(\overline{x}\colon \overline{T})\{\psi'\}[\overline{\eta}] \tag{3}$$

where m is a method name, $\overline{x}$ are the parameter names and $\overline{y}$ the names of auxiliary variables used in the pre/post conditions ψ, ψ' and in the effect $\overline{\eta}$. For (3) to be well formed, $\psi, \psi', \overline{\eta}$ should be wf in $\Gamma, \overline{y}\colon \overline{U}, \overline{x}\colon \overline{T}$. Less obviously, $\overline{\eta}$ must not contain $\mathbf{wr}\, z$ for any z in $\overline{y}, \overline{x}$; this enforces the usual constraint on value-parameters, so their use in postconditions refers to their initial value. Judgement (2) is well formed just if φ, φ', $\overline{\varepsilon}$, and Δ are well formed in Γ, and moreover $\Gamma; \mathit{sigs}(\Delta) \vdash C$ where *sigs* extracts the method signatures from a set of method specifications.

Selected proof rules are in Figs. 5 and Fig. 6.[4] There are also sound rules to delete temporary effects that are restored, e.g., if $x.f$ is written but eventually restored:

$$\text{NoUpd}\quad \frac{\{\varphi\}\ C\ \{\varphi'\}[\mathbf{wr}\langle x\rangle\bullet f, \overline{\varepsilon}] \qquad \mathbf{rd}\, x \star \overline{\varepsilon} \qquad \mathbf{rd}\, y \star \overline{\varepsilon} \qquad \varphi \vee \varphi' \Rightarrow x.f = y}{\{\varphi\}\ C\ \{\varphi'\}[\overline{\varepsilon}]}$$

N.B. that the rules are not formulated to preserve well-formedness. Rather, they are to be instantiated only with well formed antecedents and consequents. For example, in rule VAR the formulas φ, φ' cannot mention x since it is not in dom(Γ) as would be required for the consequent to be well formed.

Sometimes we distinguish between *proper rules* and *axioms* for correctness judgements; axioms have no antecedent correctness judgements (though they may have "side conditions" written above the inference line, e.g., in CALL). Unqualified, "rule" encompasses both.

In a sequence like $y\colon= \mathbf{new}\, K; y.f\colon= 0$, the effect $\mathbf{wr}\langle y\rangle\bullet f$ of the update cannot be retained as an effect of the sequence, since in the initial state of the sequence y has a

[4]For readability, we omit the type assignment to variables (Γ) in each rule where it is the same (and unconstrained) in all the antecedent and consequent correctness judgements. We also omit "$\Delta \vdash$" in most cases where the antecedents and consequents use the same Δ and impose no constraints on it.

$$\textsc{Call}\ \frac{\Delta \text{ contains } (\overline{y}{:}\,\overline{U})\{\varphi\}m(\overline{x}{:}\,\overline{T})\{\varphi'\}[\overline{\varepsilon}] \qquad \psi \equiv \varphi/\overline{x}{\to}\overline{z} \qquad \psi' \equiv \varphi'/\overline{x}{\to}\overline{z} \qquad \overline{\varepsilon}' \equiv \overline{\varepsilon}/\overline{x}{\to}\overline{z}}{\Delta \vdash \{\psi\}\ m(\overline{z})\ \{\psi'\}[\overline{\varepsilon}']}$$

$$\textsc{Var}\quad \frac{\Delta \vdash^{\Gamma,x:T} \{\varphi\}\ C\ \{\varphi'\}[\mathbf{wr}\,x, \overline{\varepsilon}]}{\Delta \vdash^{\Gamma} \{\varphi\}\ \mathbf{var}\ x{:}\ T\ \mathbf{in}\ C\ \mathbf{end}\ \{\varphi'\}[\overline{\varepsilon}]}$$

$$\textsc{Alloc}\quad \{\mathbf{true}\}\ x{:}=\mathbf{new}\,K\ \{x \neq \mathbf{null} \wedge \mathbf{type}(\langle x\rangle, K)\}[\mathbf{wr}\,x, \mathbf{wr\,all}, \mathbf{fr}\langle x\rangle]$$

$$\textsc{Seq}\ \frac{\begin{array}{c}\{\varphi\}\ C_1\ \{\psi\}[\overline{\varepsilon}_1, \mathbf{fr}\,R] \qquad \{\psi\}\ C_2\ \{\varphi'\}[\overline{\varepsilon}_2, \mathbf{wr}\,\overline{R}{\bullet}\overline{f}] \qquad \overline{\varepsilon}_1 \text{ is } \mathbf{fr}\text{-free} \\ \psi \Rightarrow R_i \subseteq R \text{ for each } \mathbf{wr}\,\overline{R_i{\bullet}f}_i \qquad \overline{\varepsilon}_2 \text{ is } \varphi/\overline{\varepsilon}_1\text{-immune} \qquad R \text{ is } \psi/(\overline{\varepsilon}_2, \mathbf{wr}\,\overline{R}{\bullet}\overline{f})\text{-immune}\end{array}}{\{\varphi\}\ C_1\,;\,C_2\ \{\varphi'\}[\overline{\varepsilon}_1, \overline{\varepsilon}_2, \mathbf{fr}\,R]}$$

$$\textsc{Bind}\quad \frac{\Delta, \Delta' \vdash^{\Gamma} \{\varphi\}\ C\ \{\varphi'\}[\overline{\varepsilon}] \qquad \begin{array}{l}\text{Each } (\overline{y}_i{:}\,\overline{U}_i)\{\psi_i\}m_i(\overline{x}_i{:}\,\overline{T}_i)\{\psi_i'\}[\overline{\varepsilon}_i] \text{ in } \Delta' \\ \text{has } \Delta, \Delta' \vdash^{\Gamma,\overline{x}_i:\overline{T}_i,\overline{y}_i:\overline{U}_i} \{\psi_i\}\ \overline{B}_i\ \{\psi_i'\}[\overline{\varepsilon}_i]\end{array}}{\Delta \vdash^{\Gamma} \{\varphi\}\ \mathbf{let}\ \mathit{sigs}(\Delta')\ \mathbf{be}\ \overline{B}\ \mathbf{in}\ C\ \{\varphi'\}[\overline{\varepsilon}]}$$

Figure 5: Selected syntax directed rules of RL.

$$\textsc{Frame}\ \frac{\{\varphi\}\ C\ \{\varphi'\}[\overline{\varepsilon}] \qquad \varphi \vdash \overline{\delta}\ \mathbf{frm}\ \psi \qquad \varphi \Rightarrow \overline{\delta} \star \overline{\varepsilon}}{\{\varphi \wedge \psi\}\ C\ \{\varphi' \wedge \psi\}[\overline{\varepsilon}]} \qquad \textsc{Assum}\ \frac{\Delta \vdash \{\varphi\}\ C\ \{\varphi'\}[\overline{\varepsilon}]}{\Delta, \Delta' \vdash \{\varphi\}\ C\ \{\varphi'\}[\overline{\varepsilon}]}$$

$$\textsc{SubEff}\ \frac{\{\varphi\}\ C\ \{\varphi'\}[\overline{\varepsilon}] \qquad \varphi \vdash \overline{\varepsilon} \leq \overline{\varepsilon}'}{\{\varphi\}\ C\ \{\varphi'\}[\overline{\varepsilon}']} \qquad \textsc{SubFr}\ \frac{\{\varphi\}\ C\ \{\varphi'\}[\overline{\varepsilon}, \mathbf{fr}\,R] \qquad \varphi' \vdash R' \subseteq R}{\{\varphi\}\ C\ \{\varphi'\}[\overline{\varepsilon}, \mathbf{fr}\,R']}$$

$$\textsc{Subst}\ \frac{\{\varphi\}\ C\ \{\varphi'\}[\overline{\varepsilon}] \qquad (\varphi/x{\to}F) \Rightarrow \mathrm{ftpt}(F) \star (\overline{\varepsilon}/x{\to}F) \qquad \mathbf{rd}\,x \notin \overline{\varepsilon} \qquad \mathbf{wr}\,x \notin \overline{\varepsilon}}{\{\varphi/x{\to}F\}\ C\ \{\varphi'/x{\to}F\}[\overline{\varepsilon}/x{\to}F]}$$

Figure 6: Selected structural rules of RL; $\varphi/x{\to}F$ denotes capture-avoiding substitution.

different value than in the initial state of the update. Rule Seq removes writes to fresh objects and uses immunity conditions to ensure that the remaining effects are sensibly propagated. Region expression R is $\varphi/\overline{\varepsilon}$-*immune* provided $\varphi \Rightarrow \mathrm{ftpt}(R) \star \overline{\varepsilon}$ is valid. Effect set $\overline{\varepsilon}_2$ is $\varphi/\overline{\varepsilon}_1$-*immune* provided that for all R, f such that $\mathbf{wr}\,R{\bullet}f$ occurs in $\overline{\varepsilon}_2$, R is $\varphi/\overline{\varepsilon}_1$-immune.

The semantics is as follows. Let t be a state transformer for Γ. Define $t \models^{\Gamma} \{\varphi\} - \{\varphi'\}[\overline{\varepsilon}]$ iff for all Γ-states σ, σ' such that $\sigma \models \varphi$ we have $t(\sigma) \neq ⋔$ and if $t(\sigma) = \sigma'$ then $\sigma' \models \varphi'$ and $\sigma \to \sigma' \models \overline{\varepsilon}$. If $\Gamma, \overline{y}{:}\,\overline{U}$ is well formed and t is a state transformer for Γ, define $t \otimes \overline{y}{:}\,\overline{U}$ to be the state transformer for $\Gamma, \overline{y}{:}\,\overline{U}$ that discards $\overline{y}$ from the initial state and applies t. If t returns a state, it gets extended with $\overline{y}$ mapped to their initial values; otherwise, the outcome ($\bot$ or ⋔) is retained. Let μ be a method environment for $\Gamma; \mathit{sigs}(\Delta)$. Define $\mu \models^{\Gamma} \Delta$ iff $\mu(m) \otimes \overline{y}{:}\,\overline{U} \models^{\Gamma,\overline{x}:\overline{T},\overline{y}:\overline{U}} \{\varphi\} - \{\varphi'\}[\overline{\varepsilon}]$ for every specification $(\overline{y}{:}\,\overline{U})\{\varphi\}m(\overline{x}{:}\,\overline{T})\{\varphi'\}[\overline{\varepsilon}]$ in Δ. Judgement $\Delta \vdash^{\Gamma} \{\varphi\}\ C\ \{\varphi'\}[\overline{\varepsilon}]$ is *valid* iff for all μ such that $\mu \models^{\Gamma} \Delta$ we have $[\![\Gamma; \mathit{sigs}(\Delta) \vdash C]\!]\mu \models^{\Gamma} \{\varphi\} - \{\varphi'\}[\overline{\varepsilon}]$.

The main result of [1] is that every rule preserves validity, whence the following.

Proposition 5.1 (RL is sound) If $\Delta \vdash^{\Gamma} \{\varphi\}\ C\ \{\varphi'\}[\overline{\varepsilon}]$ is derivable then it is valid. $\square$

6 The extended logic, EL

Recall from Sect. 2 that the antecedent of rule Frame2 is an extended judgement form that distingushes between effects of client code and effects of methods of the module of interest. The EL judgements also record the discipline by which the module invariant is protected from client effects.

The context is partitioned into the form $\Delta; \Delta^*$, where Δ and Δ^* are both sets of method specifications, with no method name in common.[5] Those in Δ are treated just as in RL; one may think of it as the interface for the ambient libraries. Those in Δ^* are associated with a designated module and are the focus of the higher order frame rule. The effect clause is partitioned into the form $\overline{\varepsilon}; \overline{\varepsilon}^*$. Effects for calls to procedures in Δ^* are put in $\overline{\varepsilon}^*$, all other effects into $\overline{\varepsilon}$. Finally, the turnstile is subscripted with a *confinement invariant* χ and *client effect bound* $\overline{\eta}$. So an *EL judgement* takes the form $\Delta; \Delta^* \vdash^{\Gamma}_{\chi;\overline{\eta}} \{\varphi\}\ C\ \{\varphi'\}[\overline{\varepsilon}; \overline{\varepsilon}^*]$. It is well formed just if $\chi, \varphi, \varphi', \overline{\eta}, \overline{\varepsilon}, \overline{\varepsilon}^*$ are well formed in Γ.

Both χ and $\overline{\eta}$ are propagated unchanged through the rules for this judgement, and are not otherwise mentioned in those rules —with the exception of Frame2 and also RLtoEL which is given below.

Validity of EL judgements is defined in terms of a corresponding RL judgement that forgets χ and $\overline{\eta}$. This is used as a sanity check on the EL rules but as noted earlier does not account for the key rule, Frame2.

Definition 6.1 (valid EL judgement) An EL judgment $\Delta; \Delta^* \vdash^{\Gamma}_{\chi;\overline{\eta}} \{\varphi\}\ C\ \{\varphi'\}[\overline{\varepsilon}; \overline{\varepsilon}^*]$ is *valid* iff the associated RL judgement $\Delta, \Delta^* \vdash^{\Gamma} \{\varphi\}\ C\ \{\varphi'\}[\overline{\varepsilon}, \overline{\varepsilon}^*]$ is.[6]

For each of the proper rules of RL except Bind —but not the axioms— there is a corresponding EL rule; see Fig. 7. There are two additional EL rules. The first converts an RL judgement to an EL judgement with arbitrarily chosen Δ^* but imposing the constraints[7] χ and $\overline{\eta}$:

$$\text{RLtoEL}\ \frac{\Delta \vdash \{\varphi\}\ C\ \{\varphi'\}[\overline{\varepsilon}] \qquad \varphi \Rightarrow \chi \qquad \varphi \vdash \overline{\varepsilon} \leq \overline{\eta}}{\Delta; \Delta^* \vdash_{\chi;\overline{\eta}} \{\varphi\}\ C\ \{\varphi'\}[\overline{\varepsilon}; \varnothing]}$$

For example, one can use the RL axiom Call followed by RLtoEL for calls to methods in Δ whereas Call* is only for methods in Δ^*.

[5] The symbol ; binds less tightly than comma. The asterisk in "Δ^*" is not a mathematical operator but merely a way to decorate the letter to obtain a fresh identifier.

[6] Here and elsewhere we refrain from mentioning that the judgements involved should be well formed.

[7] Technically, it is possible to dispense with $\chi, \overline{\eta}$ and instead thread the invariant's footprint, $\overline{\delta}$, through the EL judgements, using in RLtoEL the condition $\varphi \Rightarrow \overline{\delta} \star \overline{\varepsilon}$. But this loses the explicit separation discipline.

$$\textsc{Var*}\ \frac{\Delta;\Delta^* \vdash^{\Gamma,x:T}_{\chi;\overline{\eta}} \{\varphi\}\ C\ \{\varphi'\}[\mathbf{wr}\,x,\overline{\varepsilon};\overline{\varepsilon}^*]}{\Delta;\Delta^* \vdash^{\Gamma}_{\chi;\overline{\eta}} \{\varphi\}\ \mathbf{var}\ x\colon T\ \mathbf{in}\ C\ \mathbf{end}\ \{\varphi'\}[\overline{\varepsilon};\overline{\varepsilon}^*]}$$

$$\textsc{Call*}\ \frac{\Delta^*\text{ contains }(\overline{y}\colon \overline{U})\{\varphi\}m(\overline{x}\colon \overline{T})\{\varphi'\}[\overline{\varepsilon}] \qquad \psi \equiv \varphi/\overline{x}{\to}\overline{z} \qquad \psi' \equiv \varphi/\overline{x}{\to}\overline{z} \qquad \overline{\varepsilon}' \equiv \overline{\varepsilon}/\overline{x}{\to}\overline{z}}{\Delta;\Delta^* \vdash_{\chi;\overline{\eta}} \{\psi\}\ m(\overline{z})\ \{\psi'\}[\varnothing;\overline{\varepsilon}']}$$

$$\textsc{Frame*}\ \frac{\Delta;\Delta^* \vdash_{\chi;\overline{\eta}} \{\varphi\}\ C\ \{\varphi'\}[\overline{\varepsilon};\overline{\varepsilon}^*] \qquad \varphi \vdash \overline{\delta}\ \mathbf{frm}\ \psi \qquad \varphi \Rightarrow \overline{\delta} \star (\overline{\varepsilon},\overline{\varepsilon}^*)}{\Delta;\Delta^* \vdash_{\chi;\overline{\eta}} \{\varphi \wedge \psi\}\ C\ \{\varphi' \wedge \psi\}[\overline{\varepsilon};\overline{\varepsilon}^*]}$$

Figure 7: Selected correctness rules for EL.

The last EL rule is Frame2 (see Sect. 2). Whereas the antecedent of RLtoEL is an RL judgement and the consequent an EL judgement, the reverse is true of Frame2. The rule uses a notation from separation type systems [3] to add an invariant, ψ, to the pre- and post-conditions of some specifications. Let Δ and ψ be well formed in Γ. Define $\Delta \otimes \psi$ to be the set of method specifications $(\overline{y}\colon \overline{U})\{\varphi \wedge \psi\}m(\overline{x}\colon \overline{T})\{\varphi' \wedge \psi\}[\overline{\varepsilon}]$ such that Δ contains $(\overline{y}\colon \overline{U})\{\varphi\}m(\overline{x}\colon \overline{T})\{\varphi'\}[\overline{\varepsilon}]$.

The extended logic, EL for short, involves both RL and EL judgements. We say "EL rules" for the rules in Fig. 7 together with the special rules RLtoEL and Frame2. The logic EL consists of the RL rules together with the EL rules, as well as the rules for the subsidiary judgements already present in RL. The main result of the paper is that Frame2 is admissible. That implies soundness of EL, given the following.

Lemma 6.2 (EL rule soundness) Aside from Frame2, every EL rule is sound: if its antecedents (if any) are valid and the side conditions hold then the consequent is valid. □

For the "forgetful" semantics of Def. 6.1, Frame2 is patently unsound.[8] It is straightforward to prove the Lemma: aside from Frame2, the EL rules merely propogate the confinement invariant and client effect bound and otherwise mimic their RL counterparts.

For proving RL-judgements, there is no way to use EL-judgements or rules except as needed to use Frame2, since no other EL rule derives an RL-judgement.

In a derivation, not using Frame2, of an EL-judgement $\Delta;\Delta^* \vdash^{\Gamma}_{\chi;\overline{\eta}} \{\varphi\}\ C\ \{\varphi'\}[\overline{\varepsilon};\overline{\varepsilon}^*]$, every use of Call* is instantiated with $\chi, \overline{\eta}, \Delta^*$ (rather than some other χ', $\overline{\eta}'$, or Δ'^*), because the other EL rules preserve $\chi, \overline{\eta}, \Delta^*$ unchanged.

7 The extended logic is sound

Recall the transformation described in the first paragraph of Sect. 1. By augmenting each use of RLtoEL with a suitable use of Frame, and conjoining the invariant everywhere except in the subproofs for antecedents of RLtoEL, we can prove:

[8] For example, instantiate Frame2 with $\chi \equiv \mathbf{true}$, $\psi \equiv (x = 0)$, $C \equiv x\!:= 1$, $\varphi \equiv \mathbf{true}$, $\varphi' \equiv \mathbf{true}$, $\overline{\varepsilon} \equiv \mathbf{wr}\,x$, $\overline{\eta} = \varnothing$, and $\overline{\delta} \equiv \mathbf{rd}\,x$. Then $\overline{\delta} \star \overline{\eta}$ is valid and the rule's side conditions hold. Moreover, under the forgetful semantics the antecedent $\vdash_{\chi;\overline{\eta}} \{\mathbf{true}\}\ x\!:= 1\ \{\mathbf{true}\}[\mathbf{wr}\,x;\varnothing]$ is valid but not so the consequent $\vdash \{x = 0\}\ x\!:= 1\ \{x = 0\}[\mathbf{wr}\,x]$.

Theorem 7.1 (admissibility of Frame2) For any RL or EL judgement provable in EL, there is a proof (in EL) in which Frame2 is not used.

Proof: By induction on the given derivation D of the given judgement J, and by cases on the last rule in D. If the last rule is anything besides Frame2, the induction hypothesis yields derivations for the antecedents (if any) without using Frame2; these, followed by the last rule instance, yield a derivation of J without using Frame2.

If the last rule is Frame2, the induction hypothesis yields a derivation D' of the antecedent correctness judgement without any use of Frame2. To conclude the proof in this case, we appeal to Lemma 7.2 for D'. □

Lemma 7.2 (Frame2 elimination) Suppose judgement

$$\Delta;\Delta^* \vdash^{\Gamma}_{\chi;\overline{\eta}} \{\varphi\}\ C\ \{\varphi'\}[\overline{\varepsilon};\overline{\varepsilon}^*] \tag{4}$$

has a derivation D that does not use Frame2. Suppose that $\chi \vdash \overline{\delta}$ **frm** ψ is derivable, $\chi \Rightarrow \overline{\delta} \star \overline{\eta}$ is valid, and ψ is well formed in Γ, so that Frame2 can be instantiated with antecedent (4) to yield

$$\Delta,(\Delta^*\otimes\psi) \vdash^{\Gamma} \{\varphi\wedge\psi\}\ C\ \{\varphi'\wedge\psi\}[\overline{\varepsilon},\overline{\varepsilon}^*] \tag{5}$$

Then there is a derivation of (5) that does not use Frame2, nor any other EL rule.

The proof is by induction on D and by cases on the last rule in D.

If the last rule is Call*, then (5) can be obtained directly because the requisite specification is in $\Delta^*\otimes\psi$. In detail, since the rule's consequent is (4) the instance must look like

$$\frac{\Delta^* \text{ contains } (\overline{y}\colon\overline{U})\{\varphi_0\}m(\overline{x}\colon\overline{T})\{\varphi_0'\}[\overline{\varepsilon}_0] \qquad \varphi \equiv \varphi_0/\overline{x}\to\overline{z} \qquad \varphi' \equiv \varphi_0/\overline{x}\to\overline{z} \qquad \overline{\varepsilon}^* \equiv \overline{\varepsilon}_0/\overline{x}\to\overline{z}}{\Delta;\Delta^*\vdash_{\chi;\overline{\eta}} \{\varphi\}\ m(\overline{z})\ \{\varphi'\}[\overline{\varepsilon};\overline{\varepsilon}^*]}$$

(and $\overline{\varepsilon}$ is $\varnothing$). By definition, $\Delta^*\otimes\psi$ contains $(\overline{y}\colon\overline{U})\{\varphi_0\wedge\psi\}m(\overline{x}\colon\overline{T})\{\varphi_0'\wedge\psi\}[\overline{\varepsilon}_0]$, and thus the union $\Delta,(\Delta^*\otimes\psi)$ also contains this method specification. Since ψ and Δ^* are wf in Γ, no variable in $\overline{x}$ is in ψ, so $(\varphi_0\wedge\psi)/\overline{x}\to\overline{z} \equiv (\varphi_0/\overline{x}\to\overline{z})\wedge\psi$. Now we can get (5) for $m(\overline{z})$ by instantiating rule Call.

If the last rule is RLtoEL, the situation looks like this:

$$\frac{\dfrac{\vdots}{\Delta\vdash\{\varphi\}\ C\ \{\varphi'\}[\overline{\varepsilon}]} \qquad \varphi\Rightarrow\chi \qquad \chi\vdash\overline{\varepsilon}\le\overline{\eta}}{\Delta;\Delta^*\vdash_{\chi;\overline{\eta}}\{\varphi\}\ C\ \{\varphi'\}[\overline{\varepsilon};\varnothing]} \tag{6}$$

(and $\overline{\varepsilon}^*$ in (4) is $\varnothing$). We replace it by an instance of Frame for ψ, followed by Assum to add $\Delta^*\otimes\psi$, as follows:

$$\dfrac{\dfrac{\dfrac{\vdots}{\Delta\vdash\{\varphi\}\ C\ \{\varphi'\}[\overline{\varepsilon}]} \qquad \varphi\vdash\overline{\delta}\ \mathbf{frm}\ \psi \qquad \varphi\Rightarrow\overline{\delta}\star\overline{\varepsilon}}{\Delta\vdash\{\varphi\wedge\psi\}\ C\ \{\varphi'\wedge\psi\}[\overline{\varepsilon}]}\ \textsc{Frame}}{\Delta,(\Delta^*\otimes\psi)\vdash\{\varphi\wedge\psi\}\ C\ \{\varphi'\wedge\psi\}[\overline{\varepsilon}]}\ \textsc{Assum}$$

The side conditions for Frame are justified as follows. We have $\varphi \Rightarrow \chi$ from (6) and $\chi \vdash \overline{\delta}$ **frm** ψ by hypothesis, so we get the first side condition, $\varphi \vdash \overline{\delta}$ **frm** ψ. We have $\chi \vdash \overline{\varepsilon} \leq \overline{\eta}$ from (6) and $\chi \Rightarrow \overline{\delta} \star \overline{\eta}$ by hypothesis so we get $\chi \Rightarrow \overline{\delta} \star \overline{\varepsilon}$ by the right-monotonicity property of separators (i.e., if $\chi \Rightarrow \overline{\delta} \star \overline{\eta}$ and $\chi \vdash \overline{\varepsilon} \leq \overline{\eta}$ then $\chi \Rightarrow \overline{\delta} \star \overline{\varepsilon}$). Then using $\varphi \Rightarrow \chi$ again we get the second side condition, $\varphi \Rightarrow \overline{\delta} \star \overline{\varepsilon}$.

If the last rule of D is any other rule, it must be one of the proper rules of EL (like those in Fig. 7). For each of these rules, call it X*, the argument is similar: By induction, we obtain for each of the antecedent judgements a derivation (without Frame2) of the judgment with ψ conjoined to pre- and post-condition; this yields judgements of the form $\Delta, (\Delta^* \otimes \psi) \vdash \{\varphi \wedge \psi\}\, B\, \{\varphi' \wedge \psi\}[\overline{\varepsilon}']$ to which the RL rule X can be applied to obtain (5). In some cases a bit of extra reasoning is needed. For example, consider this derivation ending with an instance of Conj*:

$$\dfrac{\dfrac{\vdots}{\Delta; \Delta^* \vdash_{\chi;\overline{\eta}} \{\varphi_1\}\ C\ \{\varphi_1'\}[\overline{\varepsilon}; \overline{\varepsilon}^*]} \qquad \dfrac{\vdots}{\Delta; \Delta^* \vdash_{\chi;\overline{\eta}} \{\varphi_2\}\ C\ \{\varphi_2'\}[\overline{\varepsilon}; \overline{\varepsilon}^*]}}{\Delta; \Delta^* \vdash_{\chi;\overline{\eta}} \{\varphi_1 \wedge \varphi_2\}\ C\ \{\varphi_1' \wedge \varphi_2'\}[\overline{\varepsilon}; \overline{\varepsilon}^*]}\ \textsc{Conj}^*$$

By induction we get the antecedents in the derivation

$$\dfrac{\dfrac{\dfrac{\vdots}{\Delta, (\Delta^* \otimes \psi) \vdash \{\varphi_1 \wedge \psi\}\ C\ \{\varphi_1' \wedge \psi\}[\overline{\varepsilon}; \overline{\varepsilon}^*]} \qquad \dfrac{\vdots}{\Delta, (\Delta^* \otimes \psi) \vdash \{\varphi_2 \wedge \psi\}\ C\ \{\varphi_2' \wedge \psi\}[\overline{\varepsilon}; \overline{\varepsilon}^*]}}{\Delta, (\Delta^* \otimes \psi) \vdash \{\varphi_1 \wedge \psi \wedge \varphi_2 \wedge \psi\}\ C\ \{\varphi_1' \wedge \psi \wedge \varphi_2' \wedge \psi\}[\overline{\varepsilon}; \overline{\varepsilon}^*]}\ \textsc{Conj}}{\Delta, (\Delta^* \otimes \psi) \vdash \{\varphi_1 \wedge \varphi_2 \wedge \psi\}\ C\ \{\varphi_1' \wedge \varphi_2' \wedge \psi\}[\overline{\varepsilon}; \overline{\varepsilon}^*]}$$

where the last step is by Conseq using the tautology $\varphi_1 \wedge \psi \wedge \varphi_2 \wedge \psi \iff \varphi_1 \wedge \varphi_2 \wedge \psi$. In separation logic, what would be needed here is $(\varphi_1 * \psi) \wedge (\varphi_2 * \psi) \iff (\varphi_1 \wedge \varphi_2) * \psi$, which holds, e.g., if ψ is precise.

As another example, suppose we have

$$\dfrac{\dfrac{\vdots}{\Delta; \Delta^* \vdash_{\chi;\overline{\eta}} \{\varphi\}\ C\ \{\varphi'\}[\overline{\varepsilon}; \overline{\varepsilon}^*]} \qquad \varphi \vdash \overline{\delta}'\ \mathbf{frm}\ \psi' \qquad \varphi \Rightarrow \overline{\delta}' \star (\overline{\varepsilon}, \overline{\varepsilon}^*)}{\Delta; \Delta^* \vdash_{\chi;\overline{\eta}} \{\varphi \wedge \psi'\}\ C\ \{\varphi' \wedge \psi'\}[\overline{\varepsilon}; \overline{\varepsilon}^*]}\ \textsc{Frame}^*$$

We obtain $\varphi \wedge \psi \Rightarrow \overline{\delta}' \star (\overline{\varepsilon}, \overline{\varepsilon}^*)$ by logic and $\varphi \wedge \psi \vdash \overline{\delta}'$ **frm** ψ' by the consequence rule for framing (Fig. 4). By induction we obtain a derivation of the first judgement in

$$\dfrac{\dfrac{\dfrac{\vdots}{\Delta, (\Delta^* \otimes \psi) \vdash \{\varphi \wedge \psi\}\ C\ \{\varphi' \wedge \psi\}[\overline{\varepsilon}, \overline{\varepsilon}^*]} \qquad \varphi \wedge \psi \vdash \overline{\delta}'\ \mathbf{frm}\ \psi' \quad \varphi \wedge \psi \Rightarrow \overline{\delta}' \star (\overline{\varepsilon}, \overline{\varepsilon}^*)}{\Delta, (\Delta^* \otimes \psi) \vdash \{\varphi \wedge \psi \wedge \psi'\}\ C\ \{\varphi' \wedge \psi \wedge \psi'\}[\overline{\varepsilon}, \overline{\varepsilon}^*]}\ \textsc{Frame}}{\Delta, (\Delta^* \otimes \psi) \vdash \{\varphi \wedge \psi' \wedge \psi\}\ C\ \{\varphi' \wedge \psi' \wedge \psi\}[\overline{\varepsilon}, \overline{\varepsilon}^*]}\ \textsc{Conseq}$$

Corollary 7.3 (conservative extension, soundness of EL) Any RL judgement derivable in EL is derivable in RL. Any judgement derivable in EL is valid.

8 Discussion

Banerjee et al. [1] introduce a logic, essentially RL, in which command effects are specified in terms of region expressions that may depend on variables and on fields of heap objects. Their paper informally describes our second order frame rule and argues that it captures a range of separation-based disciplines for hiding module invariants in sequential object-based programs. In this paper we formalize the rule FRAME2 and the logic EL on which it is based. We show that reasoning with the rule is sound, with respect to a straightforward denotational semantics, by showing that the rule is admissible.

It is an open problem to find a semantics of EL judgements for which FRAME2 is sound as a rule. It does not seem straightforward to adapt one of the models of higher order frame rules in separation logic [3, 4]. These capture locality properties of command meanings with respect to pre-post specifications. Our logic threads the "discipline" ($\chi, \overline{\eta}$ in EL judgements) through proofs so that rule RLTOEL can impose locality properties on constituent parts of client code, allowing temporary interference with the framed invariant (e.g., via rule NOUPD). Moreover our basic frame property allows shared reads and our logic validates the rule of conjunction. Perhaps it is possible to use a standard semantics and adapt the proof techniques of O'Hearn et al. [17].[9]

An interesting feature of FRAME2 is that the framed invariant is not required to be a "precise" predicate [17] with unique semantic footprint.[10] Like in separation logic and some versions of ownership types [15], objects can be transferred across encapsulation boundaries; there is no requirement that regions grow monotonically.

Previous publications by the author and others have claimed that the second order frame rule is suited only to static modules. Indeed, a binary method like `equals` in Java already shows the need to rely on invariants of more than one object. To deal with dynamic allocation and invariants that pertain to small clusters of objects (such as a Collection object and its internal data structure, perhaps also shared by some Iterator objects), Parkinson and Bierman [2] do not use second order framing but rather make invariants everywhere explicit, but as opaque names outside the module. The Boogie methodology [12] hinges on an explicit "for all objects…" form of module invariant; this led the author to see how to use second order framing with dynamic modules, as sketched in [1]. Surely this approach can be adapted to some form of iterated separating conjunction.

Another feature of Boogie that influenced our work is that Boogie expresses encapsulation in terms of ghost state (e.g., owner pointers) and specific all-states confinement invariants. Our rule FRAME2 makes a clear distinction between a module invariant and the confinement invariant plus client effect bound used to encapsulate a module. Disciplines like Boogie and friendship [16] use ghost state to represent some dependencies; then they stipulate intermediate annotations which, according to meta-theorems, ensure invariance of the confinement and module invariants. By disentangling what is needed for Hoare's mismatch from specific ways of achieving it, we hope to make it possible to use

[9]The full version has became available in February'08 —personal communication.

[10]In practice it seems that precise predicates are common, but our rule requires no such restriction. An example imprecise predicate in RL is $\exists c\colon \mathit{Coll} \in R \mid c.\mathit{rep} \neq \mathbf{emp}$. To be specific, take $R \equiv \mathbf{all}$.

module-specific methodologies, e.g., tailored for specific design patterns, without need to bake the methodology into the verification system as in current practice.

Hoare's mismatch is subverted in object-oriented programs not only by sharing of mutable state but also by the possibility of re-entrant callbacks. Our rule FRAME2 is well suited to disciplines like Boogie [12] that deal with reentrancy using ghost state to condition invariants on whether a call is in progress. However, our rule does not capture methodologies based on "visible state semantics" [14], which deal with re-entrancy by the rather drastic requirement that each method implementation establishes *all* invariants prior to invoking *any* method.

Thanks: to Anindya Banerjee and anonymous reviewers for helpful suggestions.

References

[1] A. Banerjee, D. A. Naumann, and S. Rosenberg. Regional logic for local reasoning about global invariants. To appear in *ECOOP* 2008 (`http://www.cs.stevens.edu/~naumann/pub/rllrgi.pdf`).

[2] G. Bierman and M. Parkinson. Separation logic and abstraction. In *POPL*, 247–258, 2005.

[3] L. Birkedal, N. Torp-Smith, and H. Yang. Semantics of separation-logic typing and higher-order frame rules. In *LICS*, pages 260–269, 2005.

[4] L. Birkedal and H. Yang. Relational parametricity and separation logic. In *FoSSaCS*, pages 93–107, 2007.

[5] R. Bornat, C. Calcagno, P. W. O'Hearn, and M. J. Parkinson. Permission Accounting in Separation Logic. In *POPL*, 259–270, 2005.

[6] C. Grothoff, J. Palsberg, and J. Vitek. Encapsulating objects with confined types. *ACM TOPLAS*, 29(6), 2006.

[7] E. C. R. Hehner. Predicative programming part I. *Commun. ACM*, 27:134–143, 1984.

[8] C. A. R. Hoare. Proofs of correctness of data representations. *Acta Inf.*, 1:271–281, 1972.

[9] I. T. Kassios. Dynamic framing: Support for framing, dependencies and sharing without restriction. In *Formal Methods*, volume 4085 of *LNCS*, pages 268–283, 2006.

[10] G. T. Leavens, K. Rustan M. Leino, P. Müller. Specification and verification challenges for sequential object-oriented programs. *Formal Aspects of Computing*, 19:159–189, 2007.

[11] G. T. Leavens, D. A. Naumann, and S. Rosenberg. Preliminary definition of core JML. Stevens CS Report 2006-07.

[12] K. R. M. Leino and P. Müller. Object invariants in dynamic contexts. In *ECOOP*, pages 491–516, 2004.

[13] K. R. M. Leino and G. Nelson. Data abstraction and information hiding. *ACM TOPLAS*, 24(5):491–553, 2002.

[14] P. Müller, A. Poetzsch-Heffter, and G. T. Leavens. Modular invariants for layered object structures. *Sci. Comput. Programming*, 62(3):253–286, 2006.

[15] P. Müller and A. Rudich. Ownership transfer in Universe Types. In *OOPSLA*, 2007.

[16] D. A. Naumann and M. Barnett. Towards imperative modules: Reasoning about invariants and sharing of mutable state (extended abstract). In *LICS*, pages 313–323, 2004.

[17] P. O'Hearn, H. Yang, and J. Reynolds. Separation and information hiding. In *POPL*, pages 268–280, 2004.

[18] R. Petersen, L. Birkedal, A. Nanevski, and G. Morrisett. A realizability model of impredicative Hoare Type Theory. In *ESOP*, 2008. To appear.

[19] M. Tofte and J.-P. Talpin. Implementation of the typed call-by-value lambda-calculus using a stack of regions. In *POPL*, pages 188–201, 1994.