A Semantic Foundation for Hidden State

Jan Schwinghammer[1], Hongseok Yang[2], Lars Birkedal[3], François Pottier[4], and Bernhard Reus[5]

[1] Saarland Univ. [2] Queen Mary Univ. of London [3] IT Univ. of Copenhagen [4] INRIA [5] Univ. of Sussex

Abstract. We present the first complete soundness proof of the anti-frame rule, a recently proposed proof rule for capturing information hiding in the presence of higher-order store. Our proof involves solving a non-trivial recursive domain equation, and it helps identify some of the key ingredients for soundness.

1 Introduction

Information hiding, or *hidden state*, is one of the key design principles used by programmers in order to control the complexity of large-scale software systems. Being able to exploit this principle in the formal setting of a program logic could represent an important step towards the development of modular, scalable program verification techniques.

The idea is that an object (or function, or module) need not reveal in its interface the fact that it owns and maintains a private, mutable data structure. Hiding this internal invariant from the client has several beneficial effects. First, the complexity of the object's specification is slightly decreased. More importantly, the client is relieved from the need to thread the object's invariant through its own code. In particular, when an object has multiple clients, they are freed from the need to cooperate with one another in threading this invariant. Last, by hiding its internal state, the object escapes the restrictions on aliasing and ownership that are normally imposed to objects with mutable state.

It is worth emphasizing that *hiding* and *abstraction* (as studied, for instance, in separation logic [1–3]) are distinct mechanisms, which may co-exist within a single program logic.[1]

The recently proposed *anti-frame* proof rule [4] enables hiding in the presence of higher-order store (i.e., memory cells containing procedures or code fragments). In this paper, we study the semantic foundation of the anti-frame rule, and give the first complete soundness proof for it. Our proof involves the solution of an intricate recursive domain equation, and it helps identify some of the key ingredients for soundness.

[1] Abstraction is often implemented in terms of assertion variables (called abstract predicates by Parkinson) that describe the private data structures of an object. These variables are exposed to a client, but their definitions are not, so that the object's internals are presented to the client in an abstract form. Hiding, on the other hand, conceals the object's internals completely.

Information hiding with frame and anti-frame rules Our results push the frontier of recent logic-based approaches to information hiding. These approaches adopt a standard semantics of the programming language, and deal with information hiding on a logical basis, by extending a Hoare calculus with special proof rules. These usually take the form of *frame rules* that allow the implementation of an object to ignore (hence implicitly preserve) some of the invariants provided by the context, and of *anti-frame rules*, which allow an object to hide its internal invariant from the context [5–7, 4].

In its simplest form, the frame rule [5] states that invariants R can be added to valid triples: if $\{P\}C\{Q\}$ is valid, then so is $\{P * R\}C\{Q * R\}$, where the separating conjunction $P * R$ indicates that P and R govern disjoint regions of the heap. In subsequent developments, the rule was extended to handle higher-order procedures [6, 7] and higher-order store [8, 9]. Moreover, it was argued that both extensions of the rule support information hiding: they allow one to hide the invariant of a module [6] and to prove properties of clients, as long as the module is understood in continuation-passing style.

Thorough semantic analyses were required to determine the conditions under which these extensions of the frame rule are sound. Indeed, the soundness of these rules raises subtle issues. For instance, the frame rule for higher-order procedures turns out to be inconsistent with the conjunction rule, a standard rule of Hoare logic [6, 7]. Furthermore, seemingly innocent variants of the frame rule for higher-order store have been shown unsound [9, 10].

In the most recent development in this line of research, Pottier [4] proposed an anti-frame rule, which expresses the information hiding aspect of an object directly, instead of in continuation-passing style. Besides giving several extensive examples of how the anti-frame rule supports hidden state, Pottier argued that the anti-frame rule is sound by *sketching* a plausible syntactic argument. This argument, however, relied on several non-trivial assumptions about the existence of certain recursively defined types and recursively defined operations over types.

In this paper, we systematically study the semantic foundation of hidden state, as captured by frame and anti-frame rules, in the presence of higher-order store. In particular, we describe our soundness proof of a program logic that has both frame and anti-frame rules.

Overview of the technical development The anti-frame rule was originally proposed in an expressive type system for an ML-like language [4]. In the context of separation logic, it becomes an inference rule for deriving Hoare triples. A slightly simplified version of it takes the following form:

$$\frac{\{P \otimes R\}C\{(Q \otimes R) * R\}}{\{P\}C\{Q\}}$$

Recall that the separating conjunction $P' * Q'$ holds of a heap when the heap can be split into two sub-heaps that respectively satisfy P' and Q'. In order to specify properties of stored code, we allow assertions to contain nested triples [9], and introduce a $\otimes$ operator, whose meaning is roughly the following: $P' \otimes R'$ denotes a version of P' where R' has been $*$-conjoined with the pre- and post-conditions of every triple, including deeply nested triples.

In the anti-frame rule above, the code C can be thought of as allocating and initializing an object. The assertion R describes an internal invariant of this object, which one wishes to hide. The conjunct $- * R$ in the post-condition of the premise ensures that the invariant R is established by C, so R holds initially. The two occurrences of $- \otimes R$ in the premise guarantee that every triple that appears in the premise has R as pre- and post-conditions, so every interaction between the object and the outside preserves R. The invariant R does not appear in the conclusion of the rule, so one reasons about the rest of the program just as if the object had no internal state.

Our soundness proof of the anti-frame rule is based on two key components. The first is a new interpretation of Hoare triples, which explicates the universal and existential quantifications that are implicit in the anti-frame rule. Let $P \circ R$ abbreviate $(P \otimes R) * R$. Roughly speaking, in our interpretation, a triple$\{P\} C \{Q\}$ is valid if, for all invariants R, the triple

$$\{P \circ R\}\, C\, \{\exists R'.\; Q \circ (R \circ R')\} \tag{1}$$

holds in the standard interpretation of triples. Pottier [4] showed how the anti-frame rule allows encoding ML-like weak references in terms of strong references. Readers who are familiar with models of ML references (see, e.g., [11]) may thus find the above interpretation natural. Roughly speaking, the code C has a function type $P \to Q$, whose interpretation is: "for all worlds R, if C is given an argument of type P in world R, then, for some future world $R \circ R'$ (an extension of R), C returns a result of type Q in world $R \circ R'$."

The second element in our soundness proof is a formalization of the above intuition. Our interpretation of assertions is parameterized by a set W of *worlds*, or *invariants*, so that, semantically, an *assertion* is a function $W \to \mathcal{P}(\mathit{Heap})$ from worlds to (certain) sets of heaps. Corresponding to $R \circ R'$ in (1), there is a semantic operation $\circ$ on W that lets us combine two invariants. This operation induces a preorder on invariants, whereby $R \circ R'$ is greater than (i.e., a future world of) R.

In order to prove that the anti-frame rule is sound, we require assertions P to be monotonic with respect to the preorder on invariants: that is, $P(R)$ must be a subset of $P(R \circ R')$, for all P, R and R'. This lets us relate an assertion P at two different invariants R_0 and R_1, by first exhibiting an upper bound R_2 of R_0 and R_1 and then using the monotonicity to conclude $P(R_i) \subseteq P(R_2)$ for $i \in \{0, 1\}$. This forms an important step of our soundness proof.

In order to present our soundness proof as abstractly and elegantly as we can, we begin with an axiomatization of worlds and world composition, that is, we state a number of requirements that worlds should satisfy (Section 2). This allows us to define the interpretation of triples and establish some of its key properties in an abstract setting (Section 3).

In Sections 4 and 5, we move to a more concrete setting and present a small imperative programming language that features higher-order store, in the form of storable commands. We equip it with a proof system, which features nested Hoare triples, frame rules, and anti-frame rules. As in Pottier's original setting, in this system, it is desirable that every assertion R be allowed to play the role of an invariant. As a consequence, in this concrete instance, the set of invariants W should be isomorphic to the semantic domain of assertions.

In summary, for this instance the requirements that we have described above amount to the following non-standard recursive domain equation:

$$\mathit{Assert} \;\cong\; \mathit{Assert} \to_m \mathcal{P}(\mathit{Heap}). \tag{2}$$

The subscript $-_m$ indicates that we consider only the subset of monotonic functions. By restricting the codomain to subsets of *Heap* that satisfy particular conditions, and by further restricting the function space, we can find a solution to (a variant of) equation (2) in a category of complete metric spaces. However, because monotonicity is defined in terms of the ordering over assertions, which itself is defined in terms of the operation of composition $\circ$ on W (recall that W and *Assert* are isomorphic), we cannot do this using "off-the-shelf" techniques for solving recursive domain equations, like those of Rutten [12] or Birkedal et al. [13]. Instead, we obtain a solution by explicitly constructing the inverse limit of an appropriately chosen sequence of approximations to (2). We discuss these challenges and our solution in more detail in Section 4.

Contributions In summary, our main contributions are the following:

- We highlight which semantic ingredients are critical in establishing the validity of the frame and anti-frame rules. We hope that this will lead to increased understanding, and expect that these ingredients can be used as building blocks in the soundness proofs of future logics with information hiding principles.
- We give a proof of the soundness of a program logic that includes frame and anti-frame rules for higher-order store.

For space reasons, many proofs are omitted; some can be found in the full version of this paper [14].

2 Semantic setup

In this section, we describe semantic ingredients that can be used to validate (certain types of) anti-frame and frame rules.
Programming language. Our assumptions on the semantics of the programming language are fairly standard. We assume that there is a (pointed, chain-complete partially ordered) set of heaps, *Heap*, and that commands either diverge, terminate successfully, or fault, i.e., that $\mathit{Com} = \mathit{Heap} \multimap (\mathit{Heap} \oplus \{\mathit{error}\}_\bot)$ is the set of (strict continuous) functions into *Heap* with an error element adjoined. We also assume that there exists a family of projection functions $(\pi_k : \mathit{Heap} \multimap \mathit{Heap})_{k\in\mathbb{N}}$.

The images of these projection functions must contain only finite elements[2] and the projection functions must satisfy the following conditions:

– $\bot = \pi_0(h) \sqsubseteq \ldots \sqsubseteq \pi_k(h) \sqsubseteq \pi_{k+1}(h) \sqsubseteq \ldots \sqsubseteq h$ for all $h \in \mathit{Heap}$, i.e., the π_k's form an increasing chain of approximations of the identity on *Heap*;
– $\pi_j \circ \pi_k = \pi_{\min\{j,k\}}$ for all j, k; in particular, every π_k is idempotent;
– $\bigsqcup_k \pi_k(h) = h$, i.e., every heap is the limit of its approximations.

For example, these conditions hold if *Heap* is an SFP domain (e.g., [15]) with a particular choice of projections. Finally, we assume a partial commutative associative operation $h \cdot h'$, which intuitively lets us combine heaps with disjoint locations and is compatible with the projections: $\pi_k(h \cdot h') = \pi_k(h) \cdot \pi_k(h')$.

We write $\mathbb{N}_\infty$ for the natural numbers extended with ∞, with $\infty + k = k + \infty = \infty$. We define $\pi_\infty = \mathit{id}$ and $2^{-\infty} = 0$. The *rank of* h, written $\mathit{rnk}(h)$, is the least $k \in \mathbb{N}_\infty$ such that $\pi_k(h) = h$.

Uniformity and distance. Our program logics concern properties of heaps that are closed under the projection functions π_k. We write *UAdm* for the set of admissible[3] subsets $p \subseteq \mathit{Heap}$ that are *uniform*: for any $k \in \mathbb{N}$, if $h \in p$ then $\pi_k(h) \in p$. Using the heap combination operation, we define separating conjunction $p * q$ for $p, q \in \mathit{UAdm}$ in the usual way: $h \in p * q$ iff $h = h_1 \cdot h_2$ for some $h_1 \in p$ and $h_2 \in q$. We assume that $p * q$ is in *UAdm*.[4]

Uniformity gives rise to a notion of distance between (or, similarity of) properties of heaps. More precisely, writing $\pi_k(p)$ for the image of p under the projection π_k, the function $d(p,q) = 2^{-\sup\{k \in \mathbb{N}_\infty \mid \pi_k(p) = \pi_k(q)\}}$ defines a notion of distance on *UAdm*. This function satisfies the requirements of a *1-bounded ultrametric*: that is, d takes real values in the interval $[0,1]$, is symmetric, is such that $d(p,q) = 0$ holds iff $p = q$, and satisfies $d(p,q) \le \max\{d(p,r), d(r,q)\}$ for all $p, q, r \in \mathit{UAdm}$. With respect to this metric, *UAdm* is *complete* in the usual sense that every Cauchy sequence has a limit. The metric and this completeness result of *UAdm* make it possible to model recursively defined assertions using the Banach fixed point theorem.

Worlds and assertions. Our semantics of assertions is defined in the category *CBUlt* of complete 1-bounded ultrametric spaces and non-expansive functions. This means that every semantic domain involved in the semantics has an appropriate notion of distance and that every function is non-expansive, i.e., the distance between two outputs is no greater than the distance between the two corresponding inputs.

The main ingredients necessary for validating forms of anti-frame and frame rules are a set of possible worlds, and an interpretation of the worlds as assertions. Thus, we require:

[2] An element d in a cpo D is finite iff for all chains $\{d_n\}_{n\in\omega}$ in D, $d \sqsubseteq \bigsqcup_{n\in\omega} d_n$ implies that $d \sqsubseteq d_n$ for some n.
[3] The admissibility of p means that p is closed under limits of chains and contains $\bot$.
[4] This assumption holds when *Heap* is constructed in a standard way in terms of finite partial functions or records, just like our model in Sections 4 and 5.

1. A monoid $(W, e, \circ)$ of *worlds,* or *invariants,* where W is an object in *CBUlt* and the operation $\circ$ is non-expansive with respect to this metric. The monoid structure induces a preorder $\sqsubseteq$ on W, by $w \sqsubseteq w' \Leftrightarrow \exists w_0 \in W.\, w' = w \circ w_0$. Note that $\circ$ is in general not commutative, and that w' is obtained by extending w on the *right.* Using this preorder, we define a domain
$$\mathit{Assert} \stackrel{\mathit{def}}{=} (\tfrac{1}{2} \cdot W) \to_m \mathit{UAdm}.$$
for assertions. Here, $\frac{1}{2} \cdot W$ denotes the scaling of the distance function on W by $1/2$, and the function space consists of the non-expansive *monotone* functions. Assertions are thus parameterized by worlds, and this parameterization satisfies two conditions. The first condition is *contractiveness*, meaning that the distance between worlds gets reduced when they are used in an assertion: $d(p(w_0), p(w_1)) \leq d(w_0, w_1)/2$ for all $p \in \mathit{Assert}$ and all $w_0, w_1 \in W$. In the definition, contractiveness is formalized in two steps, first by scaling down the distance of worlds by $1/2$ and then by stipulating that assertions should preserve this scaled-down distance (i.e., be non-expansive).
The second condition is monotonicity: $p(w) \subseteq p(w \circ w_0)$ holds for all $p \in \mathit{Assert}$ and all $w, w_0 \in W$. Here, w_0 can be thought of as an invariant that is hidden in world w and revealed in world $w \circ w_0$. If some heap h satisfies the assertion p while w_0 is hidden, then h still satisfies p after w_0 is revealed. Intuitively, because the commands stored in the heap h do not know about the invariant w_0, they must preserve it.
2. A non-expansive *coercion* function $i : W \to \mathit{Assert}$, which offers a way of interpreting worlds as assertions.
We do not in general require $W \cong \mathit{Assert}$: a one-way coercion is sufficient for our purposes. In fact, it is possible to define instances of our framework where W is strictly "smaller" than Assert. Such a restriction, where not every assertion can play the role of a hidden invariant, can be exploited to establish the soundness of stronger versions of anti-frame and frame rules than the ones in the literature [4, 9]. For details, see Appendix D of the full version of this paper [14].

For the moment, we simply assume that the above ingredients are provided. In Section 4, we actually construct a particular set of worlds, together with an appropriate coercion function from worlds to assertions. In this particular case, $W \cong \mathit{Assert}$ holds.

The parameterization of assertions by a monoid W has the interesting consequence that the following $\otimes$ operator, from $\mathit{Assert} \times W$ to Assert:
$$(p \otimes w) \stackrel{\mathit{def}}{=} \lambda w_0.\, p\,(w \circ w_0)$$
is an action of this monoid over assertions, that is, it satisfies $p \otimes e = p$ and $(p \otimes w) \otimes w_0 = p \otimes (w \circ w_0)$. These are the semantic analogues of two of the distribution axioms in Pottier's type system [4], and are also included in the logic of Section 5.

Healthiness conditions. The monoid of worlds and the coercion function must satisfy two further compatibility conditions. To express these, we use the abbreviation $p \circ w \overset{def}{=} p \otimes w * i(w)$, where $*$ denotes the pointwise lifting of the separating conjunction on *UAdm* to *Assert*. The first condition is:

Condition 1. Coercions preserve $- \circ w_0$: $\forall w, w_0 \in W.\ i(w \circ w_0) = i(w) \circ w_0$.

This condition lets us explain the extension of one invariant w by a second invariant w_0 in terms of assertions. By unfolding the definition, we see that $i(w \circ w_0)$ is the assertion obtained by $*$-conjoining the assertion $i(w_0)$ to $i(w)$, and additionally ensuring that all computations described by the latter also preserve the invariant w_0.

The asymmetric nature of Condition 1 indicates that we cannot in general expect the monoid to be commutative. Instead, we require a weaker property: the existence of commutative pairs.

Definition 1 (Commutative pair). *Let w_0, w_1, a_0 and a_1 be worlds. The pair (a_0, a_1) is a* commutative pair *for (w_0, w_1) iff (1) $w_0 \circ a_1 = w_1 \circ a_0$, (2) $i(w_0) \otimes a_1 = i(a_0)$ and (3) $i(w_1) \otimes a_0 = i(a_1)$.*

If (a_0, a_1) is a commutative pair for (w_0, w_1) then $w_0 \circ a_1 = w_1 \circ a_0$ provides an upper bound of w_0 and w_1 with respect to the extension order $\sqsubseteq$. Intuitively, we can "merge" two invariants, ensuring that all computations described in the first invariant preserve the second invariant, and vice versa.

$$\begin{tikzcd}
& \cdot & \\
\cdot \arrow[ur, "a_1"] & & \cdot \arrow[ul, "a_0"'] \\
& \cdot \arrow[ul, "w_0"] \arrow[ur, "w_1"'] &
\end{tikzcd}$$

Condition 2. Every pair (w_0, w_1) of worlds has a commutative pair.

Pottier's *revelation lemma* [4], which forms the core of his sketch of a syntactic soundness argument for his anti-frame rule, assumes the existence of commutative pairs. Commutative pairs play a similarly important role in the semantic soundness proofs below. The model described in Section 4 gives a rigorous justification for their existence.

As a consequence of Condition 1 and of the fact that $\otimes$ is a monoid action, we have the following lemma:

Lemma 2. *For all $p \in Assert$ and all $w, w_0 \in W$, $(p \circ w) \circ w_0 = p \circ (w \circ w_0)$.*

3 Semantic triples, anti-frame rule and frame rules

In this section we consider the soundness of specific versions of anti-frame and frame rules, based on the semantic setting described above. For a command $c \in Com$, let $\pi_k(c)$ be the command defined by $\pi_k(c)(h) = error$ if $c(\pi_k(h)) = error$, and by $\pi_k(c)(h) = \pi_k(c(\pi_k(h)))$ if $c(\pi_k(h)) \in Heap$. Note that $\pi_\infty(c) = c$.

Definition 3. *Let tri be the ternary predicate on Assert $\times$ Com $\times$ Assert such that tri(p, c, q) holds iff*

$$\forall u \in UAdm. \forall h \in p(e) * u.\ \ c(h) \in \mathrm{Ad}(\textstyle\bigcup_w (q \circ w)(e) * u),$$

where $\mathrm{Ad}(-)$ *is the admissible downward closure.*

$$\textsc{Anti-frame}\ \frac{\models \{p \otimes w_0\}c\{q \circ w_0\}}{\models \{p\}c\{q\}} \qquad \textsc{Deep-frame}\ \frac{\models \{p\}c\{q\}}{\models \{p \circ w_0\}c\{q \circ w_0\}} \qquad \textsc{Shallow-frame}\ \frac{}{\{p\}c\{q\} \models \{p * i(w_0)\}c\{q * i(w_0)\}}$$

Fig. 1. Semantic versions of a basic form of anti-frame and frame rules

This definition deserves some explanation. First, the universal quantification over $u \in \mathit{UAdm}$ in the definition of $\mathit{tri}(p, c, q)$ "bakes in" the first-order frame rule, i.e., $\mathit{tri}(p, c, q)$ is only true of commands c that validate the first-order frame rule. Next, the existential quantification (union) over worlds $w \in W$ achieves the hiding of state from the post-condition of triples, as expressed by anti-frame rules. Because uniform admissible sets are not closed under arbitrary unions, we take the admissible downward closure. Technically, this makes sense because we assume commands are continuous and because we consider partial correctness only. We view the post-condition $\mathrm{Ad}(\bigcup_w (q \circ w)(e) * u) \subseteq \mathit{Heap}$ as a subset of $\mathit{Heap} \oplus \{\mathit{error}\}_\perp$ in the evident way. In particular, this means that $\mathit{tri}(p, c, q)$ is only true of commands c that do not fault for states in the pre-condition. This definition does not "bake in" monotonicity w.r.t. invariants (worlds): p and $q \circ w$ are "closed" just by applying them to the empty world. This is rectified in the following definition of validity:

Definition 4 (Validity). *A (semantic) triple* $\{p\}c\{q\}$ holds with respect to w and $k \in \mathbb{N}_\infty$*, which we write* $w \models_k \{p\}c\{q\}$*, iff* $\mathit{tri}(p \circ (w \circ w_0),\ \pi_k(c),\ q \circ (w \circ w_0))$ *holds for all* $w_0 \in W$*. We sometimes omit the index when* $k = \infty$*.*

As a consequence of the quantification over worlds w_0 in this definition, the validity of a triple is monotonic: if $w \models_k \{p\}c\{q\}$ and $w \sqsubseteq w'$ then $w' \models_k \{p\}c\{q\}$. The approximate validity (i.e., the case where $k \neq \infty$) is used when considering *nested* triples. Recall that assertions are the *contractive* monotone functions from W to UAdm.[5] Because nested triples will be interpreted as elements in Assert, they must be contractive. The approximations will allow us to satisfy this requirement. We write $\models \{p\}c\{q\}$ to mean that $w \models_k \{p\}c\{q\}$ for all k, w. Also, we write $\{p\}c\{q\} \models \{p'\}c'\{d'\}$ to mean that $w \models_k \{p\}c\{q\} \Rightarrow w \models_k \{p'\}c'\{q'\}$ holds for all w, k.

We are now ready to describe semantic versions of examples of anti-frame and frame rules and to prove their soundness. The semantic rules are given in Fig. 1, where the first two rules should be understood as the implication from the premise to the conclusion.

Our first lemma is a consequence of the monotonicity of assertions.

Lemma 5. *For all* w*, we have that (1)* $\mathit{tri}(p \otimes w, c, q) \Rightarrow \mathit{tri}(p, c, q)$ *and (2)* $\mathit{tri}(p, c, q \circ w) \Rightarrow \mathit{tri}(p, c, q)$*.*

[5] More precisely, they are the non-expansive monotone functions from $\frac{1}{2} \cdot W$ to UAdm.

Proof. For the first implication, suppose that $u \in \mathit{UAdm}$ and $h \in p(e) * u$. By the monotonicity of p and by $e \sqsubseteq w$ we obtain $p(e) \subseteq p(w) = p(w \circ e) = (p \otimes w)(e)$. Therefore, $h \in (p \otimes w)(e) * u$. The result now follows from the assumption that $\mathit{tri}(p \otimes w, c, q)$ holds.

For the second implication, suppose again that $u \in \mathit{UAdm}$ and $h \in p(e) * u$. We must show that $c(h) \in \mathrm{Ad}(\bigcup_{w'}(q \circ w')(e) * u)$. From the assumption that $\mathit{tri}(p, c, q \circ w)$ holds we obtain $c(h) \in \mathrm{Ad}(\bigcup_{w''}((q \circ w) \circ w'')(e) * u)$. By Lemma 2,

$$\bigcup_{w''}((q \circ w) \circ w'')(e) \;=\; \bigcup_{w''}(q \circ (w \circ w''))(e) \;\subseteq\; \bigcup_{w'}(q \circ w')(e).$$

The result then follows from the monotonicity of $*$ and the monotonicity of the closure operation $\mathrm{Ad}(\cdot)$. □

The next lemma amounts to gluing two commutative pair diagrams together (along a_0 there). Its proof involves the associativity of $\circ$ as well as Condition 1.

Lemma 6. *If (a_0, a_1) is a commutative pair for (w_0, w_1) and (b_0, a_2) is a commutative pair for (a_0, w_2) then $(b_0, a_1 \circ a_2)$ is a commutative pair for $(w_0, w_1 \circ w_2)$.*

The following proposition combines Lemmas 5 and 6, and relates the validity of two triples in our anti-frame rule.

Proposition 7. *For all worlds (w_0, w), if (a_0, a) is a commutative pair for (w_0, w), then $a \models_k \{p \otimes w_0\} c \{q \circ w_0\}$ implies $w \models_k \{p\} c \{q\}$.*

Proof. We need to show that $w \models_k \{p\} c \{q\}$, which by definition means that for all w_1, letting $w_2 = w \circ w_1$ and $d = \pi_k(c)$,

$$\mathit{tri}(p \circ w_2,\, d,\, q \circ w_2). \tag{3}$$

By Condition 2, there exists a commutative pair (b_0, a_1) for (a_0, w_1). Let $b_2 = a \circ a_1$. By Lemma 6, (b_0, b_2) is a commutative pair for $(w_0, w \circ w_1) = (w_0, w_2)$. In particular, we have $w_0 \circ b_2 = w_2 \circ b_0$ and $i(b_2) = i(w_2) \otimes b_0$. The assumed triple implies $\mathit{tri}((p \otimes w_0) \circ b_2,\, d,\, (q \circ w_0) \circ b_2)$. Thus,

$$\mathit{tri}((p \circ w_2) \otimes b_0,\, d,\, (q \circ w_2) \circ b_0) \tag{4}$$

follows, using the following equalities:

$$\begin{aligned}
(p \otimes w_0) \circ b_2 \;&=\; p \otimes (w_0 \circ b_2) * i(b_2) \;=\; p \otimes (w_2 \circ b_0) * i(w_2) \otimes b_0 \\
&=\; (p \otimes w_2) \otimes b_0 * i(w_2) \otimes b_0 \;=\; (p \otimes w_2 * i(w_2)) \otimes b_0 \\
&=\; (p \circ w_2) \otimes b_0, \\
(q \circ w_0) \circ b_2 \;&=\; q \circ (w_0 \circ b_2) \;=\; q \circ (w_2 \circ b_0) \;=\; (q \circ w_2) \circ b_0.
\end{aligned}$$

From (4), we derive the desired triple (3) as shown below:

$$\begin{aligned}
\mathit{tri}((p \circ w_2) \otimes b_0, d, (q \circ w_2) \circ b_0) &\Rightarrow \mathit{tri}(p \circ w_2, d, (q \circ w_2) \circ b_0) \\
&\Rightarrow \mathit{tri}(p \circ w_2, d, q \circ w_2).
\end{aligned}$$

Both implications hold because of Lemma 5. □

Corollary 8 (Anti-frame rule). *The anti-frame rule in Fig. 1 is sound.*

Proof. Pick w, k. Let p, c, q, w_0 be as in the anti-frame rule in Fig. 1. We must prove that $w \models_k \{p\}c\{q\}$. By Condition 2, there exists a commutative pair (a_0, a) for (w_0, w). By assumption, we have $\models \{p \otimes w_0\}c\{q \circ w_0\}$, so, in particular, $a \models_k \{p \otimes w_0\}c\{q \circ w_0\}$. By Proposition 7, this implies $w \models_k \{p\}c\{q\}$, as desired. $\square$

Next, we move on to the soundness proof of the two frame rules in Fig. 1.

Lemma 9. *The following equivalence, which expresses a distribution axiom [9], holds:* $w_0 \circ w \models_k \{p\}c\{q\}$ *iff* $w \models_k \{p \circ w_0\}c\{q \circ w_0\}$.

Proof. Pick w_1. Let $w_1' = w \circ w_1$. By Definition 4 and the associativity of $\circ$, it suffices to prove the equivalence of $tri(p \circ (w_0 \circ w_1'),\ \pi_k(c),\ q \circ (w_0 \circ w_1'))$ and $tri((p \circ w_0) \circ w_1',\ \pi_k(c),\ (q \circ w_0) \circ w_1')$. This equivalence follows from Lemma 2. $\square$

Corollary 10 (Frame rules). *The frame rules in Fig. 1 are sound.*

Proof. The soundness of the deep frame rule follows from Lemma 9. The shallow rule is sound thanks to the universal quantification over u in Definition 3. $\square$

4 A concrete model with recursively defined worlds

In this section, we consider a concrete instance of the general framework described in Sections 2 and 3 where W is isomorphic to $Assert$. The $Assert \to W$ direction of this isomorphism means that all assertions can be used as hidden invariants. This lets us define a semantic model of the program logic that is presented next (Section 5). The heap model in this particular case is given by the following recursively defined cpos:

$$Heap = Rec(Val) \quad Val = Int_\perp \oplus Com_\perp \quad Com = Heap \multimap Heap \oplus \{error\}_\perp \quad (5)$$

where $Rec(Val)$ denotes records with entries in Val labelled by positive natural numbers[6]. These labels serve as addresses or locations. The partial operation $h \cdot h'$ combines two heaps h and h' (i.e., takes the union) whenever the domains of h and h' are disjoint. When $h = \perp$ or $h' = \perp$, $h \cdot h'$ is $\perp$. The empty record provides a unit for heap combination, thus there is also a unit for the separating conjunction on $UAdm$. Finally, the solution of (5) in the category $\mathbf{Cppo}_\perp$ of pointed cpos and strict continuous functions comes equipped with a family of projections π_k that satisfy the requirements of Section 2.

The key result of this section is the following theorem:

Theorem 11. *There exists a monoid* $(W, e, \circ)$*, where* W *is an object in CBUlt with an isomorphism* ι *from* W *to* $(\frac{1}{2} \cdot W) \to_m UAdm$*. The operation* $\circ$ *satisfies*

$$\forall w_1, w_2, w \in W.\ \ \iota(w_1 \circ w_2)(w) \ = \ \iota(w_1)(w_2 \circ w) \ * \ \iota(w_2)(w).$$

[6] Formally, $Rec(D) = (\Sigma_{N \subseteq_{fin} Nats^+}(N \to D_\downarrow))_\perp$ where $N \to D_\downarrow$ is the cpo of maps from the finite address set N to $D_\downarrow = D \setminus \{\perp\}$ of non-bottom elements of D.

The equation in this theorem is just Condition 1, where the coercion i is taken to be the isomorphism ι.

Construction of the worlds W in Theorem 11. In previous work [9] we gave a model of a separation logic with nested triples and higher-order frame rules, but no anti-frame rule. For this model we needed a solution W' to the following domain equation:[7]

$$W' \;\cong\; (\tfrac{1}{2} \cdot W') \to \mathit{UAdm}. \tag{6}$$

Note that (6) is almost the same domain equation as described in Theorem 11 above, except that there is no restriction to monotonic functions in the function space on the right. One can use a general existence theorem [12, 13] to obtain a solution W' for (6) in the category *CBUlt*. In a second step, using the complete metric on W', one can then define a monoid operation $\circ$ that satisfies the equation stated in Theorem 11.

In the present setup, this two-step approach to constructing a solution $W \cong$ *Assert* and a monoid operation $\circ$ cannot be applied, however, because of the added monotonicity requirement in Theorem 11: since the order on W is defined in terms of the operation $\circ$, one needs $\circ$ already in order to *express* this equation, i.e., it appears necessary to define the operation $\circ$ *at the same time* as W. Thus we construct $W \cong (\frac{1}{2} \cdot W) \to_m \mathit{UAdm}$ explicitly, as (inverse) limit

$$W = \left\{ x \in \textstyle\prod_{k \geq 0} W_k \mid \forall k \geq 0.\ \iota_k^\circ(x_{k+1}) = x_k \right\}$$

of a sequence of "approximations" W_k of W,

$$\begin{tikzcd} W_0 \arrow[r, shift left, "\iota_0"] & W_1 \arrow[l, shift left, "\iota_0^\circ"] \arrow[r, shift left, "\iota_1"] & W_2 \arrow[l, shift left, "\iota_1^\circ"] \arrow[r, shift left, "\iota_2"] & \cdots \arrow[l, shift left, "\iota_2^\circ"] \arrow[r, shift left, "\iota_k"] & W_{k+1} \arrow[l, shift left, "\iota_k^\circ"] \arrow[r, shift left, "\iota_{k+1}"] & \cdots \arrow[l, shift left, "\iota_{k+1}^\circ"] \end{tikzcd} \tag{7}$$

Each W_k is a complete 1-bounded ultrametric space equipped with a non-expansive operation $\circ_k : W_k \times W_k \to W_k$ and a preorder $\sqsubseteq_k$, so that $W_{k+1} = (\frac{1}{2} \cdot W_k) \to_m \mathit{UAdm}$ are the non-expansive and monotone functions with respect to $\sqsubseteq_k$. The maps ι_k and ι_k° are given by $\iota_{k+1}(w) = \pi_{k+2} \circ w \circ \iota_k^\circ$ and $\iota_{k+1}^\circ(w) = \pi_{k+1} \circ w \circ \iota_k$. The diagram (7) forms a *Cauchy tower* [13], in that $\sup_w d_{k+1}(w, (\iota_k \circ \iota_k^\circ)(w))$ and $\sup_w d_k(w, (\iota_k^\circ \circ \iota_k)(w))$ become arbitrarily small as k increases. The operation $\circ_{k+1}$ is defined in terms of $\circ_k$ and ι_k°:

$$(w_1 \circ_{k+1} w_2)(w) \;\stackrel{\mathit{def}}{=}\; w_1\,(\iota_k^\circ(w_2) \circ_k w) \,*\, w_2(w).$$

One technical inconvenience is that the $\circ_k$'s are not associative. However, associativity holds "up to approximation k," $d_k((x \circ_k y) \circ_k z,\ x \circ_k (y \circ_k z)) \leq 2^{-k}$, which yields associativity "in the limit" and thus a monoid structure on W by

$$(x_k)_{k \geq 0} \circ (y_k)_{k \geq 0} \stackrel{\mathit{def}}{=} \left(\lim_{j > k} \iota_k^\circ(\ldots(\iota_{j-1}^\circ(x_j \circ_j y_j)))\right)_{k \geq 0}.$$

[7] Technically, we solved a different equation $W' \cong \frac{1}{2}(W' \to \mathit{UAdm})$. This difference is insignificant, since the solution of one equation leads to that of the other equation.

To finish the proof of Theorem 11 one shows that $\iota(w) \stackrel{\mathit{def}}{=} \lim_k(\lambda w'.\, w_{k+1}(w'_k))$ establishes an isomorphism ι between W and $(\frac{1}{2}\cdot W) \to_m \mathit{UAdm}$, which satisfies $\iota(w_1 \circ w_2) = \iota(w_1) \otimes w_2 \,*\, \iota(w_2)$ for $(p \otimes w) = \lambda w'.p(w \circ w')$.

The details of the proof are given in the full version of this paper [14].

Existence of commutative pairs. To show that W forms an instance of our semantic framework, we also need to prove the existence of commutative pairs. Given a pair (w_0, w_1) of worlds, we construct a commutative pair using properties of $\otimes$ and the coercion ι. Since the monoid operation $\otimes$ is contractive in its second argument, so is the function $f(a_0,\, a_1) = \big(\iota^{-1}(\iota(w_0) \otimes a_1),\ \iota^{-1}(\iota(w_1) \otimes a_0)\big)$ on $W \times W$. By the Banach fixed point theorem, there exists a unique pair $(a_0, a_1) = f(a_0, a_1)$. Thus $\iota(a_0) = \iota(w_0) \otimes a_1$ and $\iota(a_1) = \iota(w_1) \otimes a_0$. Since ι is injective, we can prove the remaining $w_0 \circ a_1 = w_1 \circ a_0$ as follows. For all $w \in W$,

$$\begin{aligned}
\iota(w_0 \circ a_1)(w) &= (\iota(w_0) \otimes a_1)(w) * \iota(a_1)(w) && \text{(by Theorem 11 and def. of } \otimes) \\
&= \iota(a_0)(w) * (\iota(w_1) \otimes a_0)(w) && \text{(by the above properties of } a_0,\, a_1) \\
&= \iota(a_0)(w) * \iota(w_1)(a_0 \circ w) && \text{(by def. of } \otimes) \\
&= \iota(w_1 \circ a_0)(w) && \text{(by Theorem 11).}
\end{aligned}$$

Theorem 12. *The monoid $(W, e, \circ)$ in Theorem 11 and the isomorphism ι : $W \to \mathit{Assert}$ form an instance of the framework in Section 2.*

5 Program logic

We now give one application of our semantic development. We present a program logic for higher-order store, which includes anti-frame and frame rules. Using the results of Sections 3 and 4, we define the semantics of the logic and prove its soundness.

Programming language. Fig. 2 gives the syntax of a small imperative programming language equipped with operations for stored code and heap manipulation. The expressions in the language are integer expressions, variables, and the quote expression 'C' for representing an unevaluated command C. The integer or code value denoted by expression e_1 is stored in a heap cell e_0 using $[e_0]{:=}e_1$, and this stored value is later looked up and bound to the variable y by $\mathsf{let}\ y{=}[e_0]\ \mathsf{in}\ D$. In the case that the value stored in cell e_0 is code 'C', we can run (or "evaluate") this code by executing $\mathsf{eval}\,[e_0]$. As in ML, all variables x, y, z are *immutable.* The language does not include while loops: they can be expressed by stored code (using Landin's knot). The interpretation of commands in the cpo *Com* of (5) is straightforward [9]. The interpretation of the quote operation, 'C', uses the injection of *Com* into *Val* in Section 4.

Assertions and distribution axioms. As in previous work [9], our assertion language is first-order intuitionistic logic, extended with the separating connectives $\mathsf{e}, *$, and the points-to predicate $\mapsto$ [5]. The syntax of assertions appears in Fig. 2. The standard connectives are omitted.

The most distinguishing features of the assertion language are Hoare triples $\{P\}e\{Q\}$ and invariant extensions $P \otimes Q$. The fact that a triple is an assertion

$e \in \mathit{Exp} ::=$	$-1 \mid 1 \mid e_1{+}e_2 \mid \ \ldots \ \mid x \mid `C'$	integer expression, variable, quote
$C \in \mathit{Com} ::=$	$[e_1]{:=}e_2 \mid \mathsf{let}\ y{=}[e]\ \mathsf{in}\ C \mid \mathsf{eval}\,[e]$	assignment, lookup, unquote
$\mid$	$\mathsf{let}\ x{=}\mathsf{new}\ (e_1, \ldots, e_n)\ \mathsf{in}\ C \mid \mathsf{free}\ e$	allocation, disposal
$\mid$	$\mathsf{skip} \mid C_1;C_2 \mid \mathsf{if}\ (e_1{=}e_2)\, C_1\, C_2$	no op, sequencing, conditional
$P, Q \in \mathit{Assn} ::=$	$e_1 \mapsto e_2 \mid \mathsf{e} \mid P * Q$	separating connectives
$\mid$	$\{P\}e\{Q\} \mid P \otimes Q$	Hoare triple, invariant extension
$\mid$	$X \mid \mu X.P \mid \ldots$	assertion variable, recursion

Fig. 2. Syntax of expressions, commands and assertions

$$
\begin{array}{rcll}
\{P\}e\{Q\} \otimes R & \Leftrightarrow & \{P \circ R\}e\{Q \circ R\} & \\
(\kappa x.P) \otimes R & \Leftrightarrow & \kappa x.(P \otimes R) & (\kappa \in \{\forall, \exists\}, x \notin \mathit{fv}(R)) \\
(P \otimes R) \otimes R' & \Leftrightarrow & P \otimes (R \circ R') & \\
(P \oplus Q) \otimes R & \Leftrightarrow & (P \otimes R) \oplus (Q \otimes R) & (\oplus \in \{\Rightarrow, \wedge, \vee, *\}) \\
P \otimes R & \Leftrightarrow & P & (R \text{ is } \mathsf{e} \text{ or } P \text{ is one of } \mathsf{true}, \mathsf{false}, \mathsf{e}, e \mapsto e', \ldots) \\
(\mu X.P) \otimes R & \Leftrightarrow & \mu X.(P \otimes R) & (X \notin \mathit{fv}(R))
\end{array}
$$

Fig. 3. Axioms for distributing $- \otimes R$

means that triples can be nested. Intuitively, the assertion $P \otimes Q$ denotes a version of P where every (possibly deeply nested) triple receives a copy of Q as an extra $*$-conjunct in its pre- and post-conditions. More precisely, the behaviour of the $\otimes$ operator is described by the axioms in Fig. 3, which let us distribute $\otimes$ through all the constructs of the assertion language. These axioms use the abbreviation $Q \circ R$ for $(Q \otimes R) * R$.

Assertions include assertion variables $X \in \mathcal{X}$, and can be recursively defined: the construct $\mu X.P$ binds X in P and satisfies the axiom $\mu X.P \Leftrightarrow P[X := \mu X.P]$. Not every recursive assertion is permitted: for $\mu X.P$ to be well-formed, we require that P be *formally contractive in* X. In short, this means that every free occurrence of X within P must lie either within a triple or within the second argument of a $\otimes$ construct. (We omit the straightforward inductive definition of formal contractiveness.) Semantically, this requirement ensures that $\mu X.P$ is well-defined as the unique fixed point of P, viewed as a function of X. In particular, all assertions of the form $\mu X.P \otimes X$, where X does not appear in P, are formally contractive. Pottier's applications of the anti-frame rule [4] make extensive use of assertions of this form.

The interpretation of assertions uses W in Section 4. Given such W and an environment η that maps variables x to values $\eta(x) \in \mathit{Val}$, we interpret an assertion P as a non-expansive function $[\![P]\!]_\eta : \mathit{Assert}^{\mathcal{X}} \to \mathit{Assert}$. The uniform admissible sets in UAdm, partially ordered by inclusion, form a complete Heyting algebra with a monotone commutative monoid. The domain Assert, ordered pointwise, inherits this structure (see Appendix B of the full version of this paper [14]). This is used to interpret the intuitionistic first-order fragment, $*$

$$[\![P \otimes R]\!]_{\eta,\xi} = [\![P]\!]_{\eta,\xi} \otimes \iota^{-1}([\![R]\!]_{\eta,\xi}) \qquad [\![\mu X.P]\!]_{\eta,\xi} = \mathit{fix}(\lambda q.\ [\![P]\!]_{\eta,\xi[X:=q]})$$

$$[\![\{P\}\text{'}C\text{'}\{Q\}]\!]_{\eta,\xi} = \lambda w.\,\{h \mid \mathit{rnk}(h) > 0 \ \Rightarrow\ w \models_{\mathit{rnk}(h)-1} \{[\![P]\!]_{\eta,\xi}\}\, [\![C]\!]_{\eta}\, \{[\![Q]\!]_{\eta,\xi}\}\}$$

Fig. 4. Interpretation of assertions

$$\textsc{Anti-frame}\ \frac{\Gamma; \Xi \vdash \{P \otimes R\}e\{Q \circ R\}}{\Gamma; \Xi \vdash \{P\}e\{Q\}} \qquad \textsc{Deep-frame}\ \frac{\Gamma; \Xi \vdash \{P\}e\{Q\}}{\Gamma; \Xi \vdash \{P \circ R\}e\{Q \circ R\}} \qquad \textsc{Shallow-frame}\ \frac{}{\Gamma; \Xi \vdash \{P\}e\{Q\} \Rightarrow \{P * R\}e\{Q * R\}}$$

Fig. 5. Proof rules from separation logic

and e of the assertion language. Fig. 4 shows three of the remaining cases. First, via the isomorphism ι^{-1}, we can turn any assertion $r \in \mathit{Assert}$ into an invariant $\iota^{-1}(r) \in W$ and thus interpret the invariant extension $P \otimes R$. Next, because P must be formally contractive in X, the map $q \mapsto [\![P]\!]_{\eta,\xi[X:=q]}$ on Assert is contractive in the metric sense: thus, by the Banach fixed point theorem, it has a unique fixed point. Finally, the interpretation of nested triples is in terms of semantic triples, and uses approximate validity to ensure non-expansiveness.

Proof rules. The logic derives judgements of the form $\Gamma; \Xi \vdash P$, where P is an assertion, and Γ and Ξ respectively bind variables and assertion variables. For instance, to prove that command C stores at cell 1 some code that writes 0 into cell 10, one would need to derive $\Gamma; \Xi \vdash \{1 \mapsto \_\}\text{'}C\text{'}\{1 \mapsto \{10 \mapsto \_\}\_\{10 \mapsto 0\}\}$.

The logic includes the standard proof rules for intuitionistic logic and the logic of bunched implications [16] as well as standard separation logic proof rules [9]. We do not repeat these rules here. Fig. 5 shows a version of the anti-frame rule and two versions of the frame rule: the deep frame rule (expressed in combination with the distribution axioms) and the first-order shallow frame rule (which takes the form of an axiom).

Theorem 13 (Soundness). *The interpretation of assertions is well-defined, and validates the distribution axioms of Fig. 3 and the inference rules of Fig. 5.*

6 Conclusion and Future Work

We have presented a semantic framework for studying the soundness of anti-frame and frame rules for languages with higher-order store. Moreover, we have presented a concrete instance of the framework, and used it to give the first rigorous proof of soundness of separation logic with anti-frame and frame rules for a language with higher-order store.

We are aware of other instantiations of the semantic framework, which can be used to show the soundness of stronger variants of the anti-frame and frame rules, provided the universe W of invariants is restricted. For space reasons, we have not included those instantiations in this extended abstract; they appear in the full version of the paper [14].

Future work includes lifting the results in this paper to Pottier's type-and-capability system as well as extending our soundness results to generalized versions of the anti-frame and frame rules where invariants evolve in more sophisticated ways over time [17, 18].

Acknowledgments We would like to thank Kristian Støvring and Jacob Thamsborg for helpful discussions. Partial support has been provided by FNU project 272-07-0305 "Modular reasoning about software" and EPSRC.

References

1. Parkinson, M., Bierman, G.: Separation logic and abstraction. In: POPL. (2005) 247–258
2. Biering, B., Birkedal, L., Torp-Smith, N.: BI-hyperdoctrines, higher-order separation logic, and abstraction. TOPLAS **29**(5) (2007)
3. Parkinson, M., Bierman, G.: Separation logic, abstraction and inheritance. In: POPL. (2008) 75–86
4. Pottier, F.: Hiding local state in direct style: a higher-order anti-frame rule. In: LICS. (2008) 331–340
5. Reynolds, J.C.: Separation logic: A logic for shared mutable data structures. In: LICS. (2002) 55–74
6. O'Hearn, P.W., Yang, H., Reynolds, J.C.: Separation and information hiding. In: POPL. (2004) 268–280
7. Birkedal, L., Torp-Smith, N., Yang, H.: Semantics of separation-logic typing and higher-order frame rules for Algol-like languages. LMCS **2**(5:1) (2006)
8. Birkedal, L., Reus, B., Schwinghammer, J., Yang, H.: A simple model of separation logic for higher-order store. In: ICALP. (2008) 348–360
9. Schwinghammer, J., Birkedal, L., Reus, B., Yang, H.: Nested Hoare triples and frame rules for higher-order store. In: CSL. (2009) 440–454
10. Pottier, F.: Three comments on the anti-frame rule. Unpublished note (July 2009)
11. Levy, P.B.: Possible world semantics for general storage in call-by-value. In: CSL. (2002) 232–246
12. Rutten, J.J.M.M.: Elements of generalized ultrametric domain theory. TCS **170**(1–2) (December 1996) 349–381
13. Birkedal, L., Støvring, K., Thamsborg, J.: The category-theoretic solution of recursive metric-space quations. Technical Report ITU-2009-119, IT University of Copenhagen (2009)
14. Schwinghammer, J., Yang, H., Birkedal, L., Pottier, F., Reus, B.: A semantic foundation for hidden state. Available at http://www.dcs.qmul.ac.uk/∼hyang/paper/fossacs10-full.pdf (December 2009)
15. Streicher, T.: Domain-theoretic Foundations of Functional Programming. World Scientific (2006)
16. O'Hearn, P.W., Pym, D.J.: The logic of bunched implications. Bulletin of Symbolic Logic **5**(2) (June 1999) 215–244
17. Pilkiewicz, A., Pottier, F.: The essence of monotonic state. Submitted (October 2009)
18. Pottier, F.: Generalizing the higher-order frame and anti-frame rules. Unpublished note (July 2009)