Math. Struct. in Comp. Science (2013), *vol.* 23, *pp.* 1–54. © Cambridge University Press 2012
doi:10.1017/S0960129512000035 First published online 31 August 2012

A step-indexed Kripke model of hidden state

JAN SCHWINGHAMMER[†], LARS BIRKEDAL[‡],
FRANÇOIS POTTIER[§], BERNHARD REUS[¶],
KRISTIAN STØVRING[‖] and HONGSEOK YANG[††]

[†]*Saarland University, Saarbrücken, Germany*
[‡]*IT University of Copenhagen, Copenhagen, Denmark*
[§]*INRIA Paris – Rocquencourt, Le Chesnay, France*
[¶]*University of Sussex, Brighton, United Kingdom*
[‖]*University of Copenhagen, Copenhagen, Denmark*
[††]*University of Oxford, Oxford, United Kingdom*

Received 15 July 2011; revised 2 December 2011

Frame and anti-frame rules have been proposed as proof rules for modular reasoning about programs. Frame rules allow the hiding of irrelevant parts of the state during verification, whereas the anti-frame rule allows the hiding of local state from the context.
We discuss the semantic foundations of frame and anti-frame rules, and present the first sound model for Charguéraud and Pottier's type and capability system including both of these rules. The model is a possible worlds model based on the operational semantics and step-indexed heap relations, and the worlds are given by a recursively defined metric space.
We also extend the model to account for Pottier's generalised frame and anti-frame rules, where invariants are generalised to *families* of invariants indexed over preorders. This generalisation enables reasoning about some well-bracketed as well as (locally) monotone uses of local state.

1. Introduction

Information hiding, or *hidden state*, is one of the key design principles used by programmers to control the complexity of large-scale software systems. 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 of 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 co-operate with one another in threading this invariant. Finally, by hiding its internal state, the object escapes the restrictions on aliasing and ownership that are normally imposed on objects with mutable state.

The recently proposed anti-frame proof rule (Pottier 2008) enables hiding in the presence of higher-order store, that is, memory cells containing (pointers to) procedures or code fragments. Thus, in combination with frame rules that allow the irrelevant parts of the state to be hidden during verification, the anti-frame rule can provide an important ingredient for modular, scalable program verification techniques. In the current paper, we study the semantic foundation of the anti-frame rule and give a soundness proof for it. Soundness means, in particular, that the execution of a well-typed program is safe (does not go wrong). Our proof involves an intricate recursive domain equation, and it helps identify some of the key ingredients for soundness.

1.1. *Information hiding with frame and anti-frame rules*

Our results contribute to a line of work on 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: for instance, by extending a Hoare calculus with special proof rules. These rules usually take the form of *frame rules* that allow the implementation of an object to ignore (and thus 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 (Reynolds 2002; O'Hearn *et al.* 2004; Birkedal *et al.* 2006; Pottier 2008).

It is worth emphasising the fact that *hiding* and *abstraction* (as studied, for instance, in separation logic (Parkinson and Bierman 2005; Biering *et al.* 2007; Nanevski *et al.* 2007; Parkinson and Bierman 2008)) are distinct mechanisms, which may co-exist within a single program logic. In recent program logics, 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 the object's internals are presented to the client in an abstract form. Hiding, on the other hand, conceals the object's internals completely.

In its simplest form, the frame rule (Reynolds 2002) 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 (O'Hearn *et al.* 2004; Birkedal *et al.* 2006) and higher-order store (Birkedal *et al.* 2008; Schwinghammer *et al.* 2009). Moreover, it was argued that both extensions of the rule support information hiding: they allow the hiding of the invariant of a module and the proof of properties of clients, provided the module is understood in continuation-passing style (O'Hearn *et al.* 2004).

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, which is a standard rule of Hoare logic (O'Hearn *et al.* 2004; Birkedal *et al.* 2006). Furthermore, seemingly innocent variants of the frame rule for higher-order store have been shown to be unsound (Schwinghammer *et al.* 2009; Pottier 2009b).

In the most recent development in this line of research, Pottier (Pottier 2008) 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. We will justify these assumptions in the current paper and give a complete soundness proof of Pottier's anti-frame rule.

1.2. *The current paper*

This paper is an extended version of results that were presented in two papers at the FOSSACS 2010 and FOSSACS 2011 conferences (Schwinghammer *et al.* 2010, 2011).

In the first of these papers, we presented our results on a semantic foundation for the anti-frame rule in the context of a simple WHILE language with higher-order store, using a denotational semantics of the programming language. In the second paper, we gave an alternative approach to constructing a model for the anti-frame rule and presented our results in the context of Charguéraud and Pottier's calculus of capabilities (Charguéraud and Pottier 2008), which not only features higher-order store but also higher-order functions. In this latter paper we based the model on an operational semantics of the programming language, using the discovery that the metric approach to solving recursive possible world equations works for both denotationally and operationally based models (Birkedal *et al.* 2011).

In the current paper we describe our results in the context of the calculus of capabilities, using operational semantics. We give details of both the original approach to constructing a model of the anti-frame rule from the FOSSACS 2010 paper (but adapted to operational semantics and step-indexing) and the alternative approach from the 2011 paper. We have chosen to use the capability calculus setup since Pottier has already shown how to reason about a range of applications with the anti-frame rule in this system (Pottier 2008). Moreover, Pottier has also proposed generalised versions of the frame and anti-frame rules (Pottier 2009a) for capabilities, and we show that our approach extends to these generalisations.

1.3. *Overview of the technical development*

Recently, Birkedal *et al.* (2011) developed a step-indexed model of Charguéraud and Pottier's type and capability system with higher-order frame rules, but without the anti-frame rule. This was a Kripke model in which capabilities are viewed as assertions (on heaps) that are indexed over recursively defined worlds: intuitively, these worlds are used to represent the invariants that have been added by the frame rules.

Proving soundness of the anti-frame rule requires a refinement of this idea as we need to know that additional invariants do not invalidate the invariants on local state that have been hidden by the anti-frame rule. This requirement can be formulated in terms of a monotonicity condition for the world-indexed assertions using an order on the worlds that is induced by invariant extension, that is, the addition of new invariants[†]. More precisely,

[†] The fact that ML-style untracked references can be encoded from strong references with the anti-frame rule (Pottier 2008) also indicates that a monotonicity condition is required: Kripke models of ML-style references involve monotonicity in the worlds (Birkedal *et al.* 2009; Ahmed *et al.* 2009).

in the presence of the anti-frame rule, it turns out that the recursive domain equation for the worlds involves monotone functions with respect to an order relation on worlds, and that this order is specified using the isomorphism of the recursive world solution itself. This circularity means that standard existence theorems (America and Rutten 1989), as used for the model without the anti-frame rule in Birkedal *et al.* (2011), cannot be applied to define the worlds.

In the current paper, we develop a new model of Charguéraud and Pottier's system, which can also be used to show soundness of the anti-frame rule. Moreover, we demonstrate how to extend our model to prove soundness of Pottier's *generalised* frame and anti-frame rules, which allow hiding of *families* of invariants (Pottier 2009a). The new model is a non-trivial extension of the earlier work because, as pointed out above, the anti-frame rule is the source of a circular monotonicity requirement. We present two alternative approaches that address this difficulty.

In the first approach, a solution to the recursive world equation is defined by an inverse-limit construction in a category of metric spaces; the approximants to this limit are defined simultaneously with suitably approximated order relations between worlds. This approach was originally used in Schwinghammer *et al.* (2010) for a separation logic variant of the anti-frame rule for a simple WHILE language (untyped and without higher-order functions), and with respect to a denotational semantics of the programming language. In the current paper, the metrics employed to define the recursive worlds are linked to an operational semantics of the programming language instead, using the step-indexing idea (Appel and McAllester 2001; Birkedal *et al.* 2011). While the construction is laborious, it results in a set of worlds that clearly has the required properties.

The second approach can loosely be described as a metric space analogue of Pitts' approach to relational properties of domains (Pitts 1996), and thus consists of two steps. First we consider a recursive metric space domain equation without any monotonicity requirement, and for which we obtain a solution by appealing to a standard existence theorem. Then we carve out a suitable subset of what might be called *hereditarily monotone* functions. We show how to define this recursively specified subset as a fixed point of a suitable operator. While this second construction is considerably simpler than the inverse-limit construction, the resulting subset of monotone functions is, however, not a solution to the original recursive domain equation. Hence, we must verify that the semantic constructions that are used to justify the anti-frame rule restrict in a suitable way to the recursively defined subset of hereditarily monotone functions.

We show that our techniques can be generalised by extending the model to Pottier's generalised frame and anti-frame rules (Pottier 2009a). For this extension, capabilities denote families of hereditarily monotone functions that are invariant under index reordering. The invariance property is expressed by considering a (recursively defined) partial equivalence relation on these families.

Outline of the paper

In the next section we give a brief overview of Charguéraud and Pottier's type and capability system with higher-order frame and anti-frame rules. In Section 3 we discuss

$$
\begin{aligned}
v &::= x \mid \langle\rangle \mid \mathsf{inj}^1\, v \mid \mathsf{inj}^2\, v \mid \langle v,v\rangle \mid \mathsf{fun}\, f(x) = t \mid l \\
t &::= v \mid (v\; t) \mid \mathsf{case}(v,v,v) \mid \mathsf{proj}^1\, v \mid \mathsf{proj}^2\, v \mid \mathsf{ref}\, v \mid \mathsf{get}\, v \mid \mathsf{set}\, v
\end{aligned}
$$

Fig. 1. Syntax

the requirements that the frame and anti-frame rules place on the worlds of the Kripke model. Section 4 gives some background on metric spaces, and Sections 5 and 6 present the two approaches to constructing the recursive worlds for the possible worlds model (readers not interested in the details of these constructions can safely skip Sections 5 and 6). The model is described and used to prove soundness of Charguéraud and Pottier's system in Section 7. In Section 8, we show how to extend the model to prove the soundness of the generalised frame and anti-frame rules.

2. A calculus of capabilities

Charguéraud and Pottier's calculus of capabilities uses (affine) capabilities and (unrestricted) singleton types to track aliasing and ownership properties in a high-level, ML-like programming language (Charguéraud and Pottier 2008). Capabilities describe the shape of heap data structures, much like the assertions of separation logic. By introducing static names for values, singleton types make it possible for capabilities to refer to values, including procedure arguments and results.

In the current paper, we focus on the semantic foundations of the frame and anti-frame rules. Therefore, the exact details of the type-and-capability system are less important: in this section, we will only give a brief overview of the calculus[†] – see Charguéraud and Pottier (2008), Pottier (2008; 2009a) and Pilkiewicz and Pottier (2011) for motivation concerning the design of the system and detailed examples of its use.

2.1. *Syntax and operational semantics*

The programming language that we consider is a standard call-by-value, higher-order language with general references, sum and product types, and polymorphic and recursive types. The grammar in Figure 1 gives the syntax of values and expressions, keeping close to the notation of Charguéraud and Pottier (2008). Here, the expression $\mathsf{fun}\, f(x) = t$ stands for the recursive procedure f with body t, and locations l range over a countably infinite set Loc.

The operational semantics (Figure 2) is given by a relation $(t \mid h) \longmapsto (t' \mid h')$ between configurations that consist of a (closed) expression t and a heap h. We take a heap h to be a finite map from locations to closed values. We use the notation $h \# h'$ to indicate that two heaps h, h' have disjoint domains, and we write $h \cdot h'$ for the union of two such heaps. We use Val to denote the set of closed values.

[†] We only consider a fragment of the capability calculus – in particular, we omit existential types and group regions.

$$
\begin{array}{rcll}
((\mathsf{fun}\,f(x) = t)\,v \mid h) & \longmapsto & (t[f := \mathsf{fun}\,f(x) = t, x := v] \mid h) & \\
(\mathsf{proj}^i\,\langle v_1, v_2\rangle \mid h) & \longmapsto & (v_i \mid h) & \text{for } i = 1,2 \\
(\mathsf{case}(v_1, v_2, \mathsf{inj}^i v) \mid h) & \longmapsto & (v_i\,v \mid h) & \text{for } i = 1,2 \\
(\mathsf{ref}\,v \mid h) & \longmapsto & (l \mid h \cdot [l \mapsto v]) & \text{if } l \notin \mathsf{dom}(h) \\
(\mathsf{get}\,l \mid h) & \longmapsto & (h(l) \mid h) & \text{if } l \in \mathsf{dom}(h) \\
(\mathsf{set}\,\langle l, v\rangle \mid h) & \longmapsto & (\langle\rangle \mid h[l := v]) & \text{if } l \in \mathsf{dom}(h) \\
(v\,t \mid h) & \longmapsto & (v\,t' \mid h') & \text{if } (t \mid h) \longmapsto (t' \mid h')
\end{array}
$$

Fig. 2. Operational semantics

Capabilities	$C ::= \emptyset\{\sigma : \mathsf{ref}\,\tau\} \mid C * C \mid \ldots$
Value types	$\tau ::= 1 \mid \mathsf{int} \mid \tau \times \tau \mid \tau + \tau \mid \chi \rightarrow \chi \mid [\sigma] \mid \ldots$
Computation types	$\chi ::= \tau \mid \chi * C \mid \exists \sigma.\chi \mid \ldots$
Value contexts	$\Delta ::= \varnothing \mid \Delta, x : \tau \mid \ldots$
Affine contexts	$\Gamma ::= \varnothing \mid \Gamma, x : \chi \mid \Gamma * C \mid \ldots$

Fig. 3. Capabilities and types

2.2. *Types and capabilities*

Charguéraud and Pottier's type system uses *capabilities*, *value types* and *computation types*. Figure 3 presents a subset of these (the full syntax is given in Section 7).

A capability C describes a heap property much like the assertions of separation logic. For instance, $\{\sigma : \mathsf{ref}\,\mathsf{int}\}$ asserts that σ is a *singleton region* inhabited by one valid location that contains an integer value. Here, σ is from a fixed set of region names. Group regions are omitted from this paper (see Section 9). More complex capabilities can be built using separating conjunction $C_1 * C_2$. We write $\mathit{RegNames}(C)$ for the region names appearing in capability C. Capabilities are affine: the type system ensures that they are never duplicated. This means that a capability can be regarded as a proof of *ownership* of the heap fragment it describes.

Value types τ classify values, which include base types like int, singleton types $[\sigma]$, and are closed under products and sums. Values are duplicable: in other words, a value does not have an 'owner'. Computation types χ describe the results of computations. They include all types of the form $\exists\vec{\sigma}.(\tau * C)$, which describe both the value and the heap that result from the evaluation of an expression. Arrow types (which are value types) have the form $\chi_1 \rightarrow \chi_2$ and thus, like the pre- and post-conditions of a triple in Hoare logic, make explicit which part of the heap is accessed and modified by a procedure call.

We allow recursive capabilities as well as recursive value and computation types, provided the recursive definition is formally contractive (Pierce 2002), that is, the recursion must go through one of the type constructors $+$, $\times$, $\rightarrow$, ref, or through the right-hand

$$\frac{\Delta, f : \chi_1 \to \chi_2, x : \chi_1 \Vdash t : \chi_2}{\Delta \vdash \mathsf{fun}\, f(x) = t : \chi_1 \to \chi_2} \qquad \frac{\Delta \vdash v : \chi_1 \to \chi_2 \qquad \Delta, \Gamma \Vdash t : \chi_1}{\Delta, \Gamma \Vdash (v\, t) : \chi_2}$$

$$\frac{\Gamma \Vdash v : \tau}{\Gamma \Vdash \mathsf{ref}\, v : \exists \sigma.[\sigma] * \{\sigma : \mathsf{ref}\, \tau\}} \qquad \frac{\Gamma \Vdash v : [\sigma] * \{\sigma : \mathsf{ref}\, \tau\}}{\Gamma \Vdash \mathsf{get}\, v : \tau * \{\sigma : \mathsf{ref}\, \tau\}}$$

$$\frac{\Gamma \Vdash v : ([\sigma] \times \tau_2) * \{\sigma : \mathsf{ref}\, \tau_1\}}{\Gamma \Vdash \mathsf{set}\, v : 1 * \{\sigma : \mathsf{ref}\, \tau_2\}}$$

Fig. 4. Some typing rules for values and expressions

side of the operator $\otimes$, which we introduce below (see Section 2.3). We will later see how this syntactic notion of contractiveness is justified by the model (Lemma 19).

Since Charguéraud and Pottier's system tracks aliasing, strong (that is, not necessarily type preserving) updates can be admitted. A possible type for such an update operation is $([\sigma] \times \tau_2) * \{\sigma : \mathsf{ref}\, \tau_1\} \to 1 * \{\sigma : \mathsf{ref}\, \tau_2\}$. Here, the argument to the procedure is a pair consisting of a location (named σ) and the value to be stored (whose type is τ_2), and the location is assumed to be allocated in the initial heap (and to hold a value of some type τ_1). The result of the procedure is the unit value $\langle\rangle$, but, as a side-effect, a value of type τ_2 will be stored at the location σ.

There are two typing judgements: $x_1 : \tau_1, \ldots, x_n : \tau_n \vdash v : \tau$ for values; and $x_1 : \chi_1, \ldots, x_n : \chi_n \Vdash t : \chi$ for expressions. The latter is similar to a separation logic triple where (the separating conjunction of) $\chi_1, \ldots, \chi_n$ serves as a precondition and χ as a postcondition. Since values cannot be reduced, there is no need for any preconditions and postconditions in the value typing judgement. Note that the set of value contexts is included in the set of affine contexts. Some of the inference rules that define the two typing judgements are given in Figure 4.

2.3. *Invariant extension, frame and anti-frame rules*

As in Pottier (2008), following Birkedal, Torp-Smith and Yang's approach to higher-order frame rules (Birkedal *et al.* 2006), each of the type-level syntactic categories is equipped with an *invariant extension* operation, $\cdot \otimes C$. Intuitively, this operation conjoins C to the domain and codomain of every arrow type that occurs within its left-hand argument, which means that the capability C is preserved by all procedures of this type.

This intuition is made precise by regarding capabilities and types modulo a structural equivalence that subsumes the 'distribution axioms' for $\otimes$ that are used to express generic higher-order frame rules (Birkedal *et al.* 2006). The two key cases of the structural equivalence are the distribution axioms for arrow types,

$$(\chi_1 \to \chi_2) \otimes C = (\chi_1 \otimes C * C) \to (\chi_2 \otimes C * C),$$

and for successive extensions,

$$(\chi \otimes C_1) \otimes C_2 = \chi \otimes (C_1 \circ C_2),$$

Monoid structures on capabilities

$$(\,\cdot\,) \circ C_2 \overset{def}{=} (\,\cdot \otimes C_2) * C_2 \qquad C_1 * C_2 = C_2 * C_1 \tag{1}$$

$$(C_1 \circ C_2) \circ C_3 = C_1 \circ (C_2 \circ C_3) \qquad (C_1 * C_2) * C_3 = C_1 * (C_2 * C_3) \tag{2}$$

$$C \circ \emptyset = C \qquad C * \emptyset = C \tag{3}$$

Monoid actions

$$(\cdot \otimes C_1) \otimes C_2 = \cdot \otimes (C_1 \circ C_2) \qquad \cdot \otimes \emptyset = \cdot \tag{4}$$

$$(\cdot * C_1) * C_2 = \cdot * (C_1 * C_2) \qquad \cdot * \emptyset = \cdot \tag{5}$$

Action by $*$ on affine environments

$$(\Gamma, x : \chi) * C = \Gamma, x : (\chi * C) = (\Gamma * C), x : \chi \tag{6}$$

Action by $\otimes$ on capabilities, types, and environments

$$(\cdot * \cdot) \otimes C = (\cdot \otimes C) * (\cdot \otimes C) \tag{7}$$

$$\{\sigma : \mathsf{ref}\,\tau\} \otimes C = \{\sigma : \mathsf{ref}\,(\tau \otimes C)\} \tag{8}$$

$$1 \otimes C = 1 \tag{9}$$

$$\mathsf{int} \otimes C = \mathsf{int} \tag{10}$$

$$(\tau_1 + \tau_2) \otimes C = (\tau_1 \otimes C) + (\tau_2 \otimes C) \tag{11}$$

$$(\tau_1 \times \tau_2) \otimes C = (\tau_1 \otimes C) \times (\tau_2 \otimes C) \tag{12}$$

$$(\chi_1 \to \chi_2) \otimes C = (\chi_1 \circ C) \to (\chi_2 \circ C) \tag{13}$$

$$[\sigma] \otimes C = [\sigma] \tag{14}$$

$$(\exists \sigma.\chi) \otimes C = \exists \sigma.(\chi \otimes C) \qquad \text{if } \sigma \notin \mathit{RegNames}(C) \tag{15}$$

$$\varnothing \otimes C = \varnothing \tag{16}$$

$$(\Gamma, x : \chi) \otimes C = (\Gamma \otimes C), x : (\chi \otimes C) \tag{17}$$

Fig. 5. Some axioms of the structural equivalence relation

where the derived operation $C_1 \circ C_2$ abbreviates the conjunction $(C_1 \otimes C_2) * C_2$. Figure 5 shows some of the axioms that define the structural equivalence. The operations $*$ and $\circ$ form two monoid structures on capabilities (equations 1–3). The operation $*$ and the invariant extension operation $\otimes$ are actions of these monoids (equations 4–17). The structural equivalence also includes the unfolding equations for recursive capabilities and types.

The view of capabilities as the assertions of a program logic provides some intuition for the 'shallow' and 'deep' frame rules, and for the (essentially dual) anti-frame rule given in Figure 6. As in separation logic, the frame rules can be used to add a capability C (which might assert the existence of an integer reference, say) as an invariant to a specification $\Gamma \Vdash t : \chi$. This is useful for local reasoning: if the expression t does not even know that this reference exists and contains an integer value, then it must preserve these facts. The difference between the shallow variant SHALLOW FRAME and the deep variant DEEP FRAME

$$\text{Shallow frame}\quad \frac{\Gamma \Vdash t : \chi}{\Gamma * C \Vdash t : \chi * C} \qquad \text{Deep frame (computations)}\quad \frac{\Gamma \Vdash t : \chi}{(\Gamma \otimes C) * C \Vdash t : (\chi \otimes C) * C}$$

$$\text{Deep frame (values)}\quad \frac{\Delta \vdash v : \tau}{\Delta \otimes C \vdash v : \tau \otimes C} \qquad \text{Anti-frame}\quad \frac{\Gamma \otimes C \Vdash t : (\chi \otimes C) * C}{\Gamma \Vdash t : \chi}$$

Fig. 6. Frame and anti-frame rules

is that the former adds C only at the top-level, whereas the latter also extends all arrow types nested inside Γ and χ, via $\cdot \otimes C$. While the frame rules can be used to reason about certain forms of information hiding (Birkedal *et al.* 2006), the anti-frame rule expresses a hiding principle more directly: the capability C can be removed from the specification if C is an invariant that is established by t (this is expressed by $\cdot * C$) and that is guaranteed to hold whenever control passes from t to the context or back (this is expressed by $\cdot \otimes C$).

2.4. *Example: Landin's knot*

Pottier (2008) illustrates the anti-frame rule using a number of applications. One of these is a fixed-point combinator implemented by means of 'Landin's knot', that is, using back-patching and recursion through the heap: employing the standard let notation as syntactic sugar, *fix* can be written as

$$\begin{aligned} \mathsf{fun}\,\mathit{fix}(f) = \mathsf{let}\ r &= \mathsf{ref}\,\langle\rangle \\ h &= \lambda y.(f\,(\lambda x.(\mathsf{get}\,r)\ x))\ y \\ \_ &= \mathsf{set}\,\langle r, h\rangle \\ \mathsf{in}\ h. & \end{aligned}$$

When the *fix* combinator is applied to a functional $f : (\chi_1 \to \chi_2) \to (\chi_1 \to \chi_2)$, a new reference cell r is allocated. This cell is created empty (it holds the unit value), but is shortly thereafter updated so as to hold the function h that $\mathit{fix}(f)$ will return. Circularly, the code for h refers to r: we are 'tying a knot in the store'. Subsequent calls by the client to h are safe because the cell r exists, because the code stored in r preserves the fact that r exists, and so on, *ad nauseam* (the code stored in r preserves the fact that the code stored in r preserves the fact that r exists, and so on). We see that the reason h is safe must be expressed as a recursive invariant. Here, this invariant takes the form of a recursive capability I, which satisfies the equation $I = \{\sigma : \mathsf{ref}\,((\chi_1 \to \chi_2) \otimes I)\}$, where σ is the singleton region that contains the cell denoted by the variable r. This invariant literally states that 'the cell r exists and contains code that preserves this very fact'.

Let us now briefly review how the body of *fix* is type-checked in the context $f : (\chi_1 \to \chi_2) \to (\chi_1 \to \chi_2)$, which we abbreviate as Γ. After the reference allocation expression, the variable r receives a singleton type $[\sigma]$, where σ is a fresh region variable, and the capability $\{\sigma : \mathsf{ref}\,1\}$ implicitly appears. We then examine the little 'callback' function

$\lambda x.(\mathsf{get}\, r)\ x$, and find that we have

$$(\Gamma, r : [\sigma]) \otimes I \vdash \lambda x.(\mathsf{get}\, r)\ x : (\chi_1 \to \chi_2) \otimes I.$$

That is, this function requires I (this justifies why reading r is permitted at all) and preserves I (because, according to I itself, the code found in r preserves I).

Let us now examine the application of f to this callback function. In the context $\Gamma \otimes I$, the function f has type

$$((\chi_1 \to \chi_2) \otimes I) * I \to ((\chi_1 \to \chi_2) \otimes I) * I.$$

That is, provided I holds when f is invoked, f will accept our callback function as an argument, and will preserve I. Formally, we find

$$(\Gamma, r : [\sigma]) \otimes I \vdash \lambda y.(f\ (\lambda x.(\mathsf{get}\, r)\ x))\ y : (\chi_1 \to \chi_2) \otimes I.$$

Thus, the function h has type $(\chi_1 \to \chi_2) \otimes I$. It is worth noting that at this point the invariant I does not yet hold: instead of I, for the moment, we have $\{\sigma : \mathsf{ref}\, 1\}$. However, by writing h into r, the third line of the definition of *fix* causes a strong update and establishes the invariant I. Indeed, the capability $\{\sigma : \mathsf{ref}\, 1\}$ is transformed into $\{\sigma : \mathsf{ref}\, ((\chi_1 \to \chi_2) \otimes I)\}$, which by definition is I. Formally, we have

$$(\Gamma, r : [\sigma]) \otimes I, h : (\chi_1 \to \chi_2) \otimes I * \{\sigma : \mathsf{ref}\, 1\} \Vdash \mathsf{set}\, \langle r, h \rangle : 1 * I.$$

From this, we obtain

$$(\Gamma, r : [\sigma]) \otimes I * \{\sigma : \mathsf{ref}\, 1\} \Vdash \mathsf{let}\ h = \dots\ \mathsf{in}\ h : (\chi_1 \to \chi_2) \otimes I * I,$$

which, by the structural equivalence axioms 17, 14, 9 and 6, can be written as

$$(\Gamma, r : [\sigma] * \{\sigma : \mathsf{ref}\, 1\}) \otimes I \Vdash \mathsf{let}\ h = \dots\ \mathsf{in}\ h : (\chi_1 \to \chi_2) \otimes I * I.$$

At this point, the anti-frame rule allows us to hide the fact that the function h relies on a reference cell:

$$\Gamma, r : [\sigma] * \{\sigma : \mathsf{ref}\, 1\} \Vdash \mathsf{let}\ h = \dots\ \mathsf{in}\ h : \chi_1 \to \chi_2.$$

In a realistic language design, the programmer would indicate that the anti-frame rule must be applied here, and would provide the definition of I. Pottier (2008) suggests a 'hide' construct for this purpose.

We can now conclude that in the context Γ, the body of *fix* has type $\chi_1 \to \chi_2$. As a result, *fix* itself receives the type

$$((\chi_1 \to \chi_2) \to (\chi_1 \to \chi_2)) \to (\chi_1 \to \chi_2).$$

This type does not contain any assertion about the heap: the fixed-point combinator presents a purely functional interface to the outside world. Thus, we can reason about the type safety of programs that *use* the fixed-point combinator without considering the reference cells used internally by that combinator.

2.5. *Example: locks*

The anti-frame rule imposes a strong requirement: the invariant must hold whenever control crosses the boundary between the 'inside' (where the invariant is visible) and the 'outside' (where the invariant is hidden). This requirement is tolerable when the invariant is very simple, as in Landin's knot, where the invariant expresses the existence of a single reference cell. However, when the invariant describes a more complex data structure, say, a hash table, this requirement becomes intolerable. Indeed, in this situation, the code *inside* the boundary needs to call hash table manipulation functions (*find*, and so on) that are defined *outside* the boundary. Thus, in order for a call to *find* to be well-typed, the caller is required to supply a capability for the hash table *and* for the invariant, where 'and' means separating conjunction. This is impossible: the hash table *is* part of the invariant, so we can have either the hash table or the invariant, but not both at the same time. This problem, first noted by Alexandre Pilkiewicz, is documented in Pottier (2009b).

A solution to this problem, or at least a workaround, was proposed in Pilkiewicz and Pottier (2011), which suggested using the anti-frame rule to implement a *lock* abstraction. Just as in concurrent separation logic (O'Hearn 2007; Gotsman *et al.* 2007; Hobor *et al.* 2008), a lock protects an invariant. Acquiring the lock causes the invariant to appear, seemingly out of thin air. We can then decide to break the invariant temporarily until we reach the point where we wish to release the lock. Releasing the lock requires the invariant to hold, and consumes it: the invariant disappears again into thin air. In short, locks can be viewed as a mechanism for hidden state. In concurrent separation logic, they are considered primitive, whereas, in a sequential setting, they can be implemented in terms of the anti-frame rule. In either setting, the use of locks implies that a problem might occur at runtime: a deadlock in a concurrent setting, or a failure (caused by an attempt to re-acquire the lock) in the present sequential setting. Thus, in using locks we get a somewhat weak static guarantee, but, in return, we obtain great flexibility. In the hash table example, a lock-based approach allows us to call *find* within the critical section, that is, to call *find* without first restoring the invariant.

Let the variable γ range over (affine) capabilities. In the following, we use γ to represent the invariant that the user wishes to associate with a lock. We define the type of locks, *lock* γ, through the following type abbreviation:

$$\mathit{lock}\ \gamma \equiv (1 \to 1 * \gamma) \times (1 * \gamma \to 1).$$

This type describes a pair of functions, or 'methods'. The left-hand component of the pair is the 'lock' method. Its type is $1 \to 1 * \gamma$, which means that this function takes a unit argument, produces a unit result, and, in addition, produces the capability γ. In other words, acquiring the lock causes its invariant to appear. Symmetrically, the right-hand component of the pair is the 'unlock' method. Its type is $1 * \gamma \to 1$, which means that this function requires the capability γ and consumes it.

The type *lock* γ is a value type. In other words, locks are ordinary values, and can be duplicated. The type system does not keep track of how locks are aliased, nor does it assign a unique owner to every lock. This allows locks to be used in flexible ways.

So how can we implement locks? The idea is to represent a lock as a hidden reference cell that holds a Boolean flag. The flag shows whether the lock is currently available. In the *locked* state, the invariant γ is not known to hold. In the *unlocked* state, the invariant holds: in fact, the capability γ is then conceptually stored inside the lock itself. In order to reflect this idea, we define the following abbreviation:

$$\mathit{flag}\ \gamma \equiv 1 + (1 * \gamma).$$

We write *locked* for the value $\mathsf{inj}^1\ \langle\rangle$ and *unlocked* for the value $\mathsf{inj}^2\ \langle\rangle$. At runtime, capabilities are erased, so a value of type $\mathit{flag}\ \gamma$ is just a Boolean value. However, this definition allows the type system to keep track of the fact that γ holds if and only if the flag is *unlocked*.

The untyped code for the function *newlock*, which dynamically allocates a new lock, is as follows:

```
fun $newlock(\_) =$
  let $r =$ ref $locked$ in
  let $lock(\_) =$
    let $content =$ get $r$ in
    case $content$ of
    | $unlocked \rightarrow$ set $\langle r, locked\rangle$
    | $locked \rightarrow$ fail
  and $unlock(\_) =$
    set $\langle r, unlocked\rangle$
  in $\langle lock, unlock\rangle$
```

The function *newlock* creates a new reference cell r in the state *locked*. This cell will be accessible only through the functions *lock* and *unlock*, which capture its address. The function *unlock* updates r so as to mark the lock as available. The function *lock* reads r and dynamically checks whether the lock is available. If it is, all is well, and r is updated so as to mark the lock as now unavailable. Otherwise, a fatal runtime failure occurs. This dynamic check ensures that two calls to *lock* without an intervening *unlock* will cause a failure. Finally, the function *newlock* returns the lock, which is represented by the pair $\langle lock, unlock\rangle$.

We will now give a sketch of how this code is type-checked.

We write σ for the singleton region that contains r. We wish to use the anti-frame rule to hide the existence of the cell r. This cell holds a flag, so, at first glance, it seems that the hidden invariant I should be defined by $I \equiv \{\sigma : \mathsf{ref}\,(\mathit{flag}\ \gamma)\}$. However, this definition is not quite right. As in Landin's knot (see Section 2.4), the invariant must be self-stable: it must be recursively defined by $I \equiv \{\sigma : \mathsf{ref}\,(\mathit{flag}\ \gamma)\} \otimes I$. Note that because $\otimes$ distributes over all of the type constructors involved here, I is also equal to $\{\sigma : \mathsf{ref}\,(\mathit{flag}\ (\gamma \otimes I))\}$.

We will not show in detail how the functions *lock* and *unlock* are type-checked. In short, we are able to derive the following judgement:

$$\begin{aligned} r : [\sigma] \Vdash \mathsf{let}\ lock(\_) = \ldots\ \mathsf{in}\ \langle lock, unlock\rangle :& \\ (1 * I \rightarrow 1 * (\gamma \otimes I) * I) \times (1 * (\gamma \otimes I) * I \rightarrow 1 * I).& \end{aligned}$$

This judgement states that *lock* and *unlock* require I (which allows them to access r) and preserve I. It states, furthermore, that *lock* produces $\gamma \otimes I$. Indeed, this capability is extracted out of r in the *unlocked* state, and, once r has been placed in the *locked* state, this capability does not have to be stored back into r, so it can be returned. Finally, the above judgement also states that *unlock* consumes $\gamma \otimes I$. Indeed, after writing *unlocked* to r, we must store $\gamma \otimes I$ into r in order to justify that $\{\sigma : \mathsf{ref}\,(\mathit{flag}\ (\gamma \otimes I))\}$ holds, that is, that I holds.

By the definition of *lock* γ and by the laws that govern the $\otimes$ operator, the above lengthy judgement can be summarised in a much clearer fashion as follows:

$$r : [\sigma] \Vdash \mathsf{let}\ \mathit{lock}(\_) = \ldots\ \mathsf{in}\ \langle \mathit{lock}, \mathit{unlock}\rangle : (\mathit{lock}\ \gamma) \otimes I.$$

Using the first-order frame rule, we frame I onto this judgement, and obtain

$$(r : [\sigma]) * I \Vdash \mathsf{let}\ \mathit{lock}(\_) = \ldots\ \mathsf{in}\ \langle \mathit{lock}, \mathit{unlock}\rangle : ((\mathit{lock}\ \gamma) \otimes I) * I.$$

By exploiting the definition of I, the law $[\sigma] \otimes I = [\sigma]$ and the fact that $\otimes$ distributes over $*$, the left-hand side of this judgement can be re-arranged in the following manner:

$$(r : [\sigma] * \{\sigma : \mathsf{ref}\,(\mathit{flag}\ \gamma)\}) \otimes I \Vdash \mathsf{let}\ \mathit{lock}(\_) = \ldots\ \mathsf{in}\ \langle \mathit{lock}, \mathit{unlock}\rangle :$$
$$((\mathit{lock}\ \gamma) \otimes I) * I.$$

We are now in a position to apply the anti-frame rule. The invariant I becomes hidden, and we obtain a more readable judgement:

$$r : [\sigma] * \{\sigma : \mathsf{ref}\,(\mathit{flag}\ \gamma)\} \Vdash \mathsf{let}\ \mathit{lock}(\_) = \ldots\ \mathsf{in}\ \langle \mathit{lock}, \mathit{unlock}\rangle : \mathit{lock}\ \gamma.$$

Now, the first line of code in the body of *newlock* produces precisely the binding $r : [\sigma]$ and the capability $\{\sigma : \mathsf{ref}\,(\mathit{flag}\ \gamma)\}$ for a fresh σ, so, from the above judgement, we deduce

$$\Vdash \mathsf{let}\ r = \ldots\ \mathsf{in}\ \langle \mathit{lock}, \mathit{unlock}\rangle : \mathit{lock}\ \gamma,$$

and conclude that in an empty typing environment, the function *newlock* has type $1 \to \mathit{lock}\ \gamma$. This type does not advertise any side effect: its domain and codomain are value types. Even though the capability γ is in general affine (and the type system has checked that we do not duplicate γ), a lock produced by *newlock* can be duplicated. The dynamic check within *lock* can be viewed as a dynamic mechanism for enforcing affinity, that is, a dynamic mechanism for preventing the duplication of the user invariant γ.

Our definition of *lock* γ as a pair of functions has an object-oriented flavour: by definition, locks are objects that support 'lock' and 'unlock' operations. If desired, it is easy, *a posteriori*, to make *lock* γ an abstract type, equipped with *newlock*, *lock*, and *unlock* operations. Pilkiewicz and Pottier rely on this abstract view of locks to build a hash-consing facility (Pilkiewicz and Pottier 2011).

3. Kripke semantics of frame and anti-frame rules

Our soundness proof of the frame and anti-frame rules is based on two key ideas. The first is an interpretation of arrow types that explains the universal and existential quantifications that are implicit in the anti-frame rule. Recall that $\cdot \circ C = \cdot \otimes C * C$ abbreviates the operation of combining two capabilities. Roughly speaking, an arrow type $\chi_1 \to \chi_2$ in our model consists of the procedures that have type

$$\forall C.\ \big(\chi_1 \circ C \to \exists C'.\ \chi_2 \circ (C \circ C')\big)$$

in a standard interpretation. Pottier (2008) showed how the anti-frame rule allows the encoding of ML-like weak references in terms of strong references. Readers familiar with Kripke models of ML references (see, for example, Levy (2002)) may thus find the above interpretation natural, by reading the type as *for all worlds C, if the procedure is given an argument of type χ_1 in world C, then, for some future world $C \circ C'$ (an extension of C), the procedure returns a result of type χ_2 in world $C \circ C'$.*

As indicated earlier, capabilities are like assertions in separation logic and thus describe heaps. However, to formalise the above meaning of arrow types, we need the second key idea of our model, which is that capabilities (as well as types and type contexts) are parameterised by invariants. This parameterisation will make it easy to interpret the invariant extension operation $\otimes$, as in earlier work (Schwinghammer *et al.* 2009; Birkedal *et al.* 2006). Concretely, rather than interpreting a capability C directly as a set of heaps, we interpret it as a function $[\![C]\!] : W \to Pred(Heap)$ that maps 'invariants' from W to sets of heaps. Essentially, invariant extension of $C \otimes C'$ is then interpreted by applying $[\![C]\!]$ to (the interpretation of) the given invariant C'.

Note that if we interpreted C simply as a set of heaps, the semantics would not maintain enough information to determine the meanings of different invariant extensions of C precisely.

Another intuitive argument for parameterisation is that using worlds enables benign information sharing among various parts of the program in the presence of callbacks. It is known from earlier work on object invariants that reasoning about callbacks (which we have here in the form of higher-order functions) amounts to proving properties about concurrent executions. Using world parameters, we can enable proofs about different concurrent executions to *share* information about the invariants hidden by the anti-frame rule. For instance, by interpreting a capability as a map $W \to Pred(Heap)$ or, equivalently, as a set of worlds and heaps, we ensure that a capability in a precondition records not only a set of heaps but also the shared invariants that a computation must preserve.

The question to ask now is what the set W of invariants should be. As the frame and anti-frame rules in Figure 6 suggest, invariants are in fact arbitrary capabilities, so W should be the set used to interpret capabilities. But, as we have just seen, capabilities should be interpreted as functions from W to $Pred(Heap)$. Thus, we are led to consider a Kripke model where the worlds are recursively defined: to a first approximation, we need a solution to the equation $W = W \to Pred(Heap)$.

In fact, in order to prove the soundness of the anti-frame rule, we will also need to consider a preorder on W and ensure that the interpretation of capabilities and types is monotone. This means that we should solve the equation

$$W \;=\; W \to_{mon} Pred(Heap). \tag{18}$$

The preorder $\sqsubseteq$ on W is induced by a monoid structure on W. More precisely, $w_1 \sqsubseteq w_1'$ holds if w_1' is $w_1 \circ w_2$ for some w_2 and some $\circ$ operation where the associative operation $\circ$ is required to satisfy both

$$(w_1 \circ w_2)(w) \;=\; (w_1 \otimes w_2)(w) * w_2(w) \tag{19}$$

and

$$(w_1 \otimes w_2)(w) = w_1(w_2 \circ w). \tag{20}$$

The first condition on $\circ$ reflects the definition of the syntactic operation $C_1 \circ C_2$, and the second is the semantic analogue of invariant extension.

The monotonicity condition in (18) states that

$$[\![C]\!]\,([\![C_1]\!]) \subseteq [\![C]\!]\,([\![C_1 \circ C_2]\!])$$

holds for any capability C – additional invariants (here the C_2 appearing in the combined invariant $C_1 \circ C_2$) cannot invalidate C with respect to a given invariant (here C_1). Intuitively, this property is necessary since C_1 may have been hidden by the anti-frame rule (so C_1 is not visible in the program logic) when the frame rule is applied to introduce C_2 later during the proof of a program.

Note that w_2 on the right-hand side of (20) is used both as an element in W and as a function on W. Hence, the well-formedness of the equation (20) assumes that the equation (18) is solved already. But at the same time, the monotonicity condition in (18) refers to the $\circ$ operation in (19) and (20), and it assumes the existence of the $\circ$ operation, creating circularity among all three equations. We will address this circularity in Sections 5 and 6, and construct sets of worlds W that satisfy a suitable variant of (18) using ultrametric spaces. To this end, we will recall some basic definitions and results about metric spaces in the next section.

4. Ultrametric spaces and uniform relations

This section summarises some basic notions from the theory of metric spaces, and introduces 'uniform relations', which will be used as building blocks for the interpretation in the following sections – for a less condensed introduction, see Smyth (1992) and Birkedal *et al.* (2010).

4.1. *Ultrametric spaces*

A *1-bounded ultrametric space* (X, d) is a metric space where the distance function $d : X \times X \to \mathbb{R}$ takes values in the closed interval $[0, 1]$ and satisfies the 'strong' triangle inequality $d(x, y) \leqslant \max\{d(x, z), d(z, y)\}$, for all $x, y, z \in X$. A *Cauchy sequence* is a sequence $(x_n)_{n\in\mathbb{N}}$ of elements in X such that for every $k \in \mathbb{N}$ there exists an index n, and for all $n_1, n_2 \geqslant n$, $d(x_{n_1}, x_{n_2}) \leqslant 2^{-k}$. A metric space is *complete* if every Cauchy sequence $(x_n)_{n\in\mathbb{N}}$ has a limit $\lim_n x_n$. A subset of a complete metric space is *closed* if it is closed under the limit operation.

A function $f : X_1 \to X_2$ between metric spaces (X_1, d_1), (X_2, d_2) is *non-expansive* if $d_2(f(x), f(y)) \leqslant d_1(x, y)$ for all $x, y \in X_1$. It is *contractive* if there exists some $\delta < 1$ such that $d_2(f(x), f(y)) \leqslant \delta \cdot d_1(x, y)$ for all $x, y \in X_1$. By the Banach fixed-point theorem, every contractive function $f : X \to X$ on a complete and non-empty metric space (X, d) has a (unique) fixed point. By multiplying the distances of (X, d) by a non-negative factor $\delta < 1$, we obtain a new ultrametric space, $\delta \cdot (X, d) = (X, d')$ where $d'(x, y) = \delta \cdot d(x, y)$; this can be used to ensure the contractiveness of functions.

The complete, 1-bounded, non-empty, ultrametric spaces and non-expansive functions between them form a Cartesian closed category **CBUlt**. The terminal object **1** is given by any singleton space, and products are given by the set-theoretic product, where the distance is the maximum of the componentwise distances. The exponential $(X_1, d_1) \to (X_2, d_2)$ has the set of non-expansive functions from (X_1, d_1) to (X_2, d_2) as underlying set, and the distance function is given by the sup metric:

$$d_{X_1 \to X_2}(f, g) = \sup\{d_2(f(x), g(x)) \mid x \in X_1\}$$

(note that the supremum always exists since it is taken over a bounded set in $\mathbb{R}$).

The notation $x \overset{n}{=} y$ means that $d(x, y) \leqslant 2^{-n}$. Each relation $\overset{n}{=}$ is an equivalence relation because of the ultrametric inequality, and we refer to this relation as 'n-equality'. Since the distances are bounded by 1, $x \overset{0}{=} y$ always holds, and the n-equalities become finer as n increases. If $x \overset{n}{=} y$ holds for all n, then $x = y$; this observation allows us to prove equalities by induction on n.

If X is *bisected*, that is, if all distances in X are of the form 2^{-n} for some n, then a function $f : X \to Y$ is non-expansive if and only if $x \overset{n}{=} x'$ implies $f(x) \overset{n}{=} f(x')$. All the metric spaces we consider in the following have this property.

4.2. *Uniform relations*

In order to rephrase the (informal) requirement (18) in **CBUlt**, we consider uniform relations (Birkedal *et al.* 2011) rather than arbitrary predicates on *Heap*. More generally, let $(A, \leqslant)$ be a preordered set. An *(upwards closed) uniform relation* on A is a subset $p \subseteq \mathbb{N} \times A$ that is downwards closed in the first and upwards closed in the second component:

$$(k, a) \in p \ \wedge \ j \leqslant k \ \wedge \ a \leqslant b \ \Rightarrow \ (j, b) \in p.$$

We write $URel(A)$ for the set of all such relations on A, and for $k \in \mathbb{N}$, we define

$$p_{[k]} = \{(j, a) \in p \mid j < k\}.$$

Note that $p_{[k]} \in URel(A)$ if $p \in URel(A)$, and that $p \subseteq p'$ implies $p_{[n]} \subseteq p'_{[n]}$. We equip $URel(A)$ with the distance function

$$d(p, q) = \inf\{2^{-n} \mid p_{[n]} = q_{[n]}\},$$

which makes $(URel(A), d)$ an object of **CBUlt**. Moreover, $URel(A)$ forms a complete Heyting algebra.

Proposition 1. *URel*(A), ordered by inclusion, forms a complete Heyting algebra. Meets and joins are given by set-theoretic intersections and unions, respectively, and implication $p \Rightarrow q$ is given by the uniform relation such that $(k, a) \in (p \Rightarrow q)$ holds if and only if for all $j \leqslant k$ and all $b \geqslant a$, we have $(j, b) \in p$ implies $(j, b) \in q$.

Meets, joins and implication are non-expansive operations on $URel(A)$ with respect to the distance function d defined above.

In our model, we use $URel(A)$ with the following concrete instances for the preorder $(A, \leqslant)$:

(1) *Heaps* $(Heap, \leqslant)$, where $h \leqslant h'$ if and only if $h' = h \cdot h_0$ for some $h_0 \# h$.
(2) *Values* $(Val, \leqslant)$, where $v \leqslant v'$ if and only if $v = v'$.
(3) *Stateful values* $(Val \times Heap, \leqslant)$, where $(v, h) \leqslant (v', h')$ if and only if $v = v'$ and $h \leqslant h'$.
(4) *Stateful expressions* $(Exp \times Heap, \leqslant)$, where $(t, h) \leqslant (t', h')$ if and only if $t = t'$ and $h = h'$.

We will also use variants of (2) and (3) where the set Val is replaced by the set of value substitutions, Env.

Proposition 2. $URel(Heap)$ forms a complete BI algebra (Pym *et al.* 2004). The separating conjunction and separating implication are given by

$$\begin{aligned}(k, h) \in (p_1 * p_2) &\Leftrightarrow \exists h_1, h_2.\ h = h_1 \cdot h_2 \ \wedge\ (k, h_1) \in p_1 \wedge\ (k, h_2) \in p_2 \\ (k, h) \in (p \twoheadrightarrow q) &\Leftrightarrow \forall j \leqslant k.\ \forall h' \# h.\ (j, h') \in p \ \Rightarrow\ (j, h \cdot h') \in q,\end{aligned}$$

and the unit for $*$ is given by $I = \mathbb{N} \times Heap$.

Up to the natural number indexing, $URel(Heap)$ is just the standard intuitionistic (in the sense that it is not 'tight') heap model of separation logic (Reynolds 2002). Both separating conjunction and separating implication are non-expansive operations on $URel(Heap)$.

In the following, we will not need to use all of the algebraic structure on uniform relations when interpreting types and capabilities. Nevertheless, the fact that the uniform relations form a complete Heyting (BI) algebra suggests that Charguéraud and Pottier's system could, in principle, be extended to a full-blown program logic by including all of the logical connectives of separation logic assertions in the syntax of capabilities.

4.3. *Preordered metric spaces*

The uniform relations $URel(A)$, ordered by inclusion, form an example of a preordered metric space. More generally, a *preordered, complete, 1-bounded ultrametric space* is an object $(X, d) \in \mathbf{CBUlt}$ equipped with a preorder $\leqslant$ such that for all Cauchy sequences $(x_n)_{n \in \mathbb{N}}$ and $(y_n)_{n \in \mathbb{N}}$, if $x_n \leqslant y_n$ holds for all $n \in \mathbb{N}$, then $\lim_n x_n \leqslant \lim_n y_n$.

For preordered, complete, 1-bounded ultrametric spaces X_1 and X_2, we write $X_1 \to_{mon} X_2$ for the set of non-expansive and monotone functions between X_1 and X_2. When equipped with the sup-distance,

$$d(f, g) = \sup\{d_2(f\,x, g\,x) \mid x \in X_1\},$$

the set $X_1 \to_{mon} X_2$ becomes an object of $\mathbf{CBUlt}$.

Proposition 3. For any preordered, complete, 1-bounded ultrametric spaces X and Y, if Y is a complete Heyting algebra with non-expansive algebra operations, then so is $X \to_{mon} Y$ when this function space is equipped with the pointwise order. Meets and joins are given by the pointwise extension of the corresponding operations on Y, and $f \Rightarrow g$ is defined by

$$(f \Rightarrow g)(x) = \bigwedge_{x' \geqslant x} \big(f(x') \Rightarrow g(x')\big).$$

In the case where Y is $URel(Heap)$, we have $X \to_{mon} URel(Heap)$ is a complete BI algebra where $*$ and $-\!\!*$ are non-expansive operations. The separating conjunction $f * g$ and its unit I are defined pointwise, and the separating implication $f -\!\!* g$ is defined by

$$(f -\!\!* g)(x) = \bigwedge_{x' \geqslant x} \big(f(x') -\!\!* g(x')\big).$$

5. Monotone recursive worlds

In this section we prove the following existence theorem.

Theorem 4 (Existence of monotone recursive worlds). There exists a preordered monoid $(W, \sqsubseteq, \circ, e)$ where W is an object of **CBUlt** with a non-expansive isomorphism ι from $\big(\frac{1}{2} \cdot W \to_{mon} URel(Heap)\big)$ to W, such that the following conditions hold:

(1) The preorder on W is given by $w \sqsubseteq w' \Leftrightarrow \exists w_0.\ w' = w \circ w_0$.
(2) The operation $\circ : W \times W \to W$ is non-expansive.
(3) For all $w_1, w_2, w \in W$,

$$\iota^{-1}(w_1 \circ w_2)(w) \;=\; \iota^{-1}(w_1)(w_2 \circ w) \;*\; \iota^{-1}(w_2)(w).$$

In condition (3), the operation $*$ is the separating conjunction on uniform heap relations described in Proposition 2. By Proposition 3, $\frac{1}{2} \cdot W \to_{mon} URel(Heap)$ is a complete BI algebra.

This theorem asserts the existence of a suitable set of worlds for the interpretation of the capability calculus. In particular, if we define

$$(f \otimes w_1)(w) = f(w_1 \circ w)$$

for $f \in \big(\frac{1}{2} \cdot W \to_{mon} URel(Heap)\big)$ and $w_1, w \in W$, we have

$$\iota^{-1}(w_1 \circ w_2)(w) \;=\; (\iota^{-1}(w_1) \otimes w_2)(w) \;*\; \iota^{-1}(w_2)(w).$$

Apart from the insertion of the isomorphism in this equation, the differences between the statement of the theorem and the informal requirements given in equations (18)–(20) are: the use of uniform relations instead of arbitrary predicates over heaps; the restriction to non-expansive functions; and the scaling factor $\frac{1}{2}$. (Note that a non-expansive function $\frac{1}{2} \cdot W \to_{mon} URel(Heap)$ is the same as a contractive function $W \to_{mon} URel(Heap)$ with contraction factor $\frac{1}{2}$.)

We prove Theorem 4 by constructing $W \cong \frac{1}{2} \cdot W \to_{mon} URel(Heap)$ explicitly, as the (inverse) limit

$$W = \big\{x \in \textstyle\prod_{k \geqslant 0} W_k \mid \forall k \geqslant 0.\ x_k = \epsilon_k^{\circ}(x_{k+1})\big\}$$

of a sequence of 'approximations' W_k of W,

$$\begin{tikzcd} W_0 \arrow[r, shift left, "\epsilon_0"] & W_1 \arrow[l, shift left, "\epsilon_0^\circ"] \arrow[r, shift left, "\epsilon_1"] & W_2 \arrow[l, shift left, "\epsilon_1^\circ"] \arrow[r, shift left, "\epsilon_2"] & \cdots \arrow[l, shift left, "\epsilon_2^\circ"] \arrow[r, shift left, "\epsilon_k"] & W_{k+1} \arrow[l, shift left, "\epsilon_k^\circ"] \arrow[r, shift left, "\epsilon_{k+1}"] & \cdots \arrow[l, shift left, "\epsilon_{k+1}^\circ"] \end{tikzcd} \tag{21}$$

Each W_k is a complete, 1-bounded, ultrametric space with distance function d_k, and comes equipped with a non-expansive operation $\circ_k : W_k \times W_k \to W_k$ and a preorder $\sqsubseteq_k$. This sequence will be defined inductively so that $W_{k+1} = \frac{1}{2} \cdot W_k \to_{mon} URel(Heap)$ consists of the non-expansive and monotone functions with respect to $\sqsubseteq_k$.

5.1. *Cauchy tower of approximants*

We define preordered, complete, 1-bounded ultrametric spaces $(W_k, \sqsubseteq_k)$, binary operations $\circ_k$ on W_k, and functions

$$\begin{tikzcd} W_k \arrow[r, shift left, "\epsilon_k"] & \left(\frac{1}{2} \cdot W_k \to_{mon} URel(Heap)\right) \arrow[l, shift left, "\epsilon_k^\circ"] \end{tikzcd}$$

by induction on k as follows:

— For $k = 0$:
  - $W_0 = \{\star\}$ is a one-point space.
  - $\circ_0$ is given by $\star \circ_0 \star = \star$.
  - $\sqsubseteq_0$ is the trivial order, $\star \sqsubseteq_0 \star$.
  - $\epsilon_0(w) = I$, the unit of the BI algebra structure on $\frac{1}{2} \cdot W_0 \to_{mon} URel(Heap)$.
  - $\epsilon_0^\circ(f) = \star$.

— For $k \geqslant 0$:
  - $W_{k+1} = \frac{1}{2} \cdot W_k \to_{mon} URel(Heap)$.
  - $\circ_{k+1}$ is given by $(f \circ_{k+1} g)(w) = f((\epsilon_k^\circ g) \circ_k w) * g(w)$.
  - $f \sqsubseteq_{k+1} g$ holds if $g \overset{k+1}{=} f \circ_{k+1} f_0$ for some $f_0 \in W_{k+1}$.
  - $\epsilon_{k+1}(f)$ sends $g \in \frac{1}{2} \cdot W_{k+1}$ to $(f(\epsilon_k^\circ g))_{[k+2]} \in URel(Heap)$.
  - $\epsilon_{k+1}^\circ(F)$ sends $w \in \frac{1}{2} \cdot W_k$ to $(F(\epsilon_k w))_{[k+1]} \in URel(Heap)$.

In the rest of this subsection we show that $\sqsubseteq_k$ indeed defines a preorder on W_k, and that ϵ_k and ϵ_k° are non-expansive and map into the space of non-expansive and monotone functions. One technical inconvenience in these proofs is that the operations $\circ_k$ are not preserved by ϵ_k and ϵ_{k-1}°, and that they are not associative. However, associativity holds 'up to approximation k', which explains the definition of $\sqsubseteq_k$ above.

Lemma 5 (Well-definedness). For all $k \geqslant 0$:

(1) ϵ_k and ϵ_k° are non-expansive functions between W_k and $\frac{1}{2} \cdot W_k \to URel(Heap)$.
(2) For all $w, w' \in W_k$, we have $w \circ_k w' \in W_k$.
(3) $\circ_k$ is a non-expansive operation on W_k.
(4) For all $w, w', w'' \in W_k$, we have $(w \circ_k w') \circ_k w'' \overset{k}{=} w \circ_k (w' \circ_k w'')$.
(5) For all $w \in W_{k+1}$, we have $I \circ_{k+1} w = w$ and $w \circ_{k+1} I \overset{k+1}{=} w$, where I is the unit of the BI algebra structure on $\frac{1}{2} \cdot W_k \to_{mon} URel(Heap)$.
(6) The relation $\sqsubseteq_k$ is a preorder on W_k.

(7) For all $w \in W_k$ and $F \in W_{k+2}$, we have $\epsilon_k(w)$ and $\epsilon^\circ_{k+1}(F)$ are monotone functions $\frac{1}{2} \cdot W_k \to_{mon} URel(Heap)$.

(8) For all $w \in W_k$, we have $\epsilon^\circ_k(\epsilon_k\, w) \overset{k}{=} w$. For all $g \in W_{k+1}$, we have $\epsilon_k(\epsilon^\circ_k\, g) \overset{k}{=} g$.

(9) For all $w, w' \in W_k$, we have $\epsilon_k(w \circ_k w') \overset{k}{=} (\epsilon_k\, w) \circ_{k+1} (\epsilon_k\, w')$. For all $g, g' \in W_{k+1}$, we have $\epsilon^\circ_k(g \circ_{k+1} g') \overset{k}{=} (\epsilon^\circ_k\, g) \circ_k (\epsilon^\circ_k\, g')$.

Proof. The properties are proved simultaneously by induction on k. The case $k = 0$ follows directly from the definitions. We will just give the key ideas for the case $k > 0$:

(1) The claimed non-expansiveness properties are consequences of the non-expansiveness of ϵ_{k-1} and ϵ°_{k-1}, which is obtained from the induction hypothesis, and from the non-expansiveness of function composition.

(2) By the induction hypothesis, $\circ_{k-1}$ and ϵ°_{k-1} are non-expansive functions; the non-expansiveness of $w \circ_k w'$ then follows with the non-expansiveness of $*$. The monotonicity of $w \circ_k w'$ follows from the definition of $\sqsubseteq_k$, the approximate associativity of $\circ_{k-1}$ being given by part (4) of the induction hypothesis, and the monotonicity of $*$ on $URel(Heap)$.

(3) For the non-expansiveness of $\circ_k$, note that $(\epsilon^\circ_{k-1}\, w_2) \circ_{k-1} w \overset{n}{=} (\epsilon^\circ_{k-1}\, w'_2) \circ_{k-1} w$ holds for all $w \in W_{k-1}$ and all $w_2, w'_2 \in W_k$ with $w_2 \overset{n}{=} w'_2$ by parts (1) and (3) of the induction hypothesis. Thus, for all w_1, w'_1 with $w_1 \overset{n}{=} w'_1$, the non-expansiveness of $*$, w_1 and w'_1 yields $(w_1 \circ_k w_2)(w) \overset{n}{=} (w'_1 \circ_k w'_2)(w)$. Since w is chosen arbitrarily, the sup-metric on W_k shows $w_1 \circ_k w_2 \overset{n}{=} w'_1 \circ_k w'_2$.

(4) Given any $x \in \frac{1}{2} \cdot W_{k-1}$, we have

$$((w \circ_k w') \circ_k w'')(x) \overset{k}{=} (w \circ_k (w' \circ_k w''))(x)$$

follows from parts (4) and (9) of the induction hypothesis. Thus, the claim follows using the sup-metric on W_k.

(5) The fact that $I \circ_{k+1} w = I$ follows easily from the definition of I. For the second claim, first note that $(\epsilon^\circ_k I) \overset{k}{=} I$ holds in W_k. Thus, $(\epsilon^\circ_k I) \circ_k x \overset{k+1}{=} I \circ_k x = I$ holds in $\frac{1}{2} \cdot W_k$ for any x by the non-expansiveness of $\circ_k$ and the first claim. From this observation, the $k+1$-equivalence of $w \circ_{k+1} I$ and w follows from the non-expansiveness of w and the definition of $\circ_{k+1}$.

(6) The fact that $\sqsubseteq_k$ is a preorder follows from its definition using parts (4) and (5).

(7) The fact that $\epsilon_k(w)(w_1) \subseteq \epsilon_k(w)(w_2)$ holds for any $w, w_1, w_2 \in W_k$ with $w_1 \sqsubseteq_k w_2$ is a consequence of the monotonicity of w, part (9) of the induction hypothesis and the definition of ϵ_k. For the second claim, note that whenever $w_1 \sqsubseteq_k w_2$, the definition of $\sqsubseteq_k$ and part (9) of the induction hypothesis yield $\epsilon_k(w_2) \overset{k}{=} \epsilon_k(w_1) \circ_{k+1} \epsilon_k(w_0)$ for some w_0. Hence,

$$\begin{aligned}
\epsilon^\circ_{k+1}(F)(w_2) &= (F(\epsilon_k\, w_2))_{[k+1]} \\
&= (F(\epsilon_k(w_1) \circ_{k+1} \epsilon_k(w_0)))_{[k+1]} \\
&\supseteq (F(\epsilon_k\, w_1))_{[k+1]} \\
&= \epsilon^\circ_{k+1}(F)(w_1)
\end{aligned}$$

by the contractiveness and monotonicity of F and the definition of ϵ°_{k+1}.

(8) The claims follow from part (8) of the induction hypothesis using the fact that function composition is non-expansive and that functions in W_k for $k > 0$ are contractive with contraction factor $\frac{1}{2}$, due to the scaling in the definition of W_k.

(9) By part (9) of the induction hypothesis and property (8),

$$\epsilon_{k-1}^{\circ}(\epsilon_k^{\circ}(\epsilon_k\, w') \circ_k w_0) \overset{k-1}{=} \epsilon_{k-1}^{\circ}(w') \circ_{k-1} \epsilon_{k-1}^{\circ}(w_0)$$

holds for all $w', w_0 \in W_k$. Thus, using the definitions of $\circ_k$ and ϵ_k, we get the contractiveness of $w \in W_k$ and the non-expansiveness of $*$,

$$\begin{aligned}(\epsilon_k\, w \circ_{k+1} \epsilon_k\, w')(w_0) &= (w(\epsilon_{k-1}^{\circ}(\epsilon_k^{\circ}(\epsilon_k\, w') \circ_k w_0)))_{[k+1]} * (w'(\epsilon_{k-1}^{\circ}\, w_0))_{[k+1]} \\ &\overset{k}{=} (w(\epsilon_{k-1}^{\circ}(w') \circ_{k-1} \epsilon_{k-1}^{\circ}(w_0)) \;*\; w'(\epsilon_{k-1}^{\circ}\, w_0))_{[k+1]} \\ &= (w \circ_k w')(\epsilon_{k-1}^{\circ}\, w_0)_{[k+1]},\end{aligned}$$

which is just $\epsilon_k(w \circ_k w')(w_0)$. Since this approximate equality holds for all w_0, the claim follows by definition of the sup-metric on W_k.

The second claim is proved similarly. □

Lemma 5(8) states that diagram (21) forms a Cauchy tower (Birkedal *et al.* 2010), meaning that $\sup_w d_{k+1}(w, \epsilon_k(\epsilon_k^{\circ}\, w))$ as well as $\sup_w d_k(w, \epsilon_k^{\circ}(\epsilon_k\, w))$ become arbitrarily small as k increases. This ensures that

$$W = \{x \in \textstyle\prod_{k \geqslant 0} W_k \mid \forall k \geqslant 0.\ x_k = \epsilon_k^{\circ}(x_{k+1})\},$$

equipped with the sup-distance, is an object of **CBUlt**. Limits of Cauchy chains in W are given componentwise.

5.2. *Monoid structure on the inverse limit*

For all $0 \leqslant k < l$, we define the functions $\epsilon_{k,l} : W_k \to W_l$ and $\epsilon_{k,l}^{\circ} : W_l \to W_k$ by

$$\begin{aligned}\epsilon_{k,l} &= \epsilon_{l-1} \cdot \ldots \cdot \epsilon_{k+1} \cdot \epsilon_k \\ \epsilon_{k,l}^{\circ} &= \epsilon_k^{\circ} \cdot \epsilon_{k+1}^{\circ} \cdot \ldots \cdot \epsilon_{l-1}^{\circ},\end{aligned}$$

which are non-expansive by Lemma 5(1).

Next, we equip W with an operation $\circ : W \times W \to W$ defined by

$$(x_k)_{k\geqslant 0} \circ (y_k)_{k \geqslant 0} \;=\; \big(\lim_{j>k} \epsilon_{k,j}^{\circ}(x_j \circ_j y_j)\big)_{k\geqslant 0}.$$

Note that the limits exist:

$$\begin{aligned}\epsilon_j^{\circ}(x_{j+1} \circ_{j+1} y_{j+1}) &\overset{j}{=} \epsilon_j^{\circ}(x_{j+1}) \circ_j \epsilon_j^{\circ}(y_{j+1}) \\ &= x_j \circ_j y_j\end{aligned}$$

by Lemma 5(9), so $(\epsilon_{k,j}^{\circ}(x_j \circ_j y_j))_{j>k}$ forms a Cauchy sequence in W_k by the non-expansiveness of $\epsilon_{k,j}^{\circ}$. Moreover, we have

$$\epsilon_k^{\circ}(\lim_{j>k+1} \epsilon_{k+1,j}^{\circ}(x_j \circ_j y_j)) = \lim_{j>k+1} \epsilon_{k,j}^{\circ}(x_j \circ_j y_j),$$

which shows that $x \circ y$ is a sequence in W. We also define $e \overset{def}{=} (e_k)_{k\geqslant 0} \in W$ by $e_k = I$.

Lemma 6. $(W, \circ, e)$ is a monoid with non-expansive multiplication $\circ$.

Proof. From the definitions of e and $\circ$, we have $e \circ w = w \circ e = w$ for all $w \in W$. To see the associativity of $\circ$, suppose $x, y, z \in W$. Lemma 5(4) shows for all j that

$$(x_j \circ_j y_j) \circ_j z_j \overset{j}{=} x_j \circ_j (y_j \circ_j z_j).$$

We then obtain

$$\begin{aligned}
x_j \circ_j (y \circ z)_j &= x_j \circ_j (\lim_{l>j} \epsilon^{\circ}_{j,l}(y_l \circ_l z_l)) \\
&= \lim_{l>j} x_j \circ_j \epsilon^{\circ}_{j,l}(y_l \circ_l z_l)) && \text{(by non-expansiveness of } \circ_j) \\
&\overset{j}{=} \lim_{l>j} x_j \circ_j (\epsilon^{\circ}_{j,l}(y_l) \circ_j \epsilon^{\circ}_{j,l}(z_l)) && \text{(by Lemma 5(9))} \\
&= \lim_{l>j} x_j \circ_j (y_j \circ_j z_j) && \text{(since } y, z \in W) \\
&\overset{j}{=} \lim_{l>j} (x_j \circ_j y_j) \circ_j z_j && \text{(by Lemma 5(4))} \\
&\overset{j}{=} (x \circ y)_j \circ_j z_j.
\end{aligned}$$

Thus, for any real number $\varepsilon > 0$, there exists $n \geqslant 0$ sufficiently large such that

$$\forall j \geqslant n.\ d_{W_j}((x \circ y)_j \circ_j z_j,\ x_j \circ_j (y \circ z)_j) < \varepsilon.$$

Since $\epsilon^{\circ}_{k,j}$ is non-expansive, this yields for all k

$$\begin{aligned}
((x \circ y) \circ z)_k &= \lim_{j>k}\ \epsilon^{\circ}_{k,j}((x \circ y)_j \circ_j z_j) \\
&= \lim_{j>k}\ \epsilon^{\circ}_{k,j}(x_j \circ_j (y \circ z)_j) \\
&= (x \circ (y \circ z))_k,
\end{aligned}$$

which proves $(x \circ y) \circ z = x \circ (y \circ z)$.

Finally, $\circ$ is non-expansive since each $\circ_j$ and $\epsilon^{\circ}_{k,j}$ is non-expansive. □

5.3. *Isomorphism between W and monotone functions on W*

As shown in Lemma 6, $\circ$ is associative and has e as a unit. Therefore, we can consider the induced preorder on W,

$$w \sqsubseteq w' \iff \exists w_0.\ w' = w \circ w_0.$$

We still need to establish an isomorphism $W \cong \frac{1}{2} \cdot W \rightarrow_{mon} URel(Heap)$ in **CBUlt** (where the monotonicity refers to this preorder $\sqsubseteq$ on W) that satisfies condition (3) in Theorem 4.

To this end, we first note that if $w' = w \circ w''$, then $w'_k \overset{k}{=} w_k \circ_k w''_k$ for all k, and, therefore, we obtain

$$\forall w, w' \in W.\ w \sqsubseteq w' \ \Rightarrow\ \forall k.\ w_k \sqsubseteq_k w'_k. \tag{22}$$

Now note that for each k and for all sequences $(w_k)_{k\geqslant 0}$ and $(w'_k)_{k\geqslant 0}$ in W we have

$$\begin{aligned} w_{k+1}(w'_k) &= \epsilon^{\circ}_{k+1}(w_{k+2})(\epsilon^{\circ}_k\, w'_{k+1}) \\ &= (w_{k+2}(\epsilon_k(\epsilon^{\circ}_k\, w'_{k+1})))_{[k+1]} \\ &\overset{k+1}{=} w_{k+2}(w'_{k+1}) \end{aligned}$$

by Lemma 5(8) and the contractiveness of w_{k+2}. Hence, $(\lambda w'.w_{k+1}(w'_k))_{k\geqslant 0}$ is a Cauchy sequence in $\frac{1}{2}\cdot W \to URel(Heap)$. In fact, it is a sequence in the (complete) subspace of monotone maps, by (22) and the fact that each w_k is monotone, so this sequence has a limit in $\frac{1}{2}\cdot W \to_{mon} URel(Heap)$. We may thus define

$$\imath^{\bullet}(w) \;=\; \lim_k(\lambda w' \in W.\, w_{k+1}(w'_k)).$$

For $g \in \frac{1}{2}\cdot W \to_{mon} URel(Heap)$, we define $\imath(g)_k \in W_k$ through the following two cases:

$$\begin{aligned} \imath(g)_0 &= \star \\ \imath(g)_{k+1} &= \lambda w \in \tfrac{1}{2}\cdot W_k.\, \big(g\big(\lim_{l>\max\{i,k\}}\; \epsilon^{\circ}_{i,l}(\epsilon_{k,l}\, w)\big)_{i\geqslant 0}\big)_{[k+1]}. \end{aligned}$$

For this definition, we first check that the sequence $(\lim_{l>\max\{i,k\}}\; \epsilon^{\circ}_{i,l}(\epsilon_{k,l}\, w))_{i\geqslant 0}$ is an element of W, so that g can be applied, and then that each $\imath(g)_{k+1}$ is monotone. To see this, let $w_1 \sqsubseteq_k w_2$, so, by the definition of $\sqsubseteq_k$, there exists $w_0 \in W_k$ such that w_2 and $w_1 \circ_k w_0$ are k-equivalent in W_k; we must show that $(\imath\, g)_{k+1}(w_1) \subseteq (\imath\, g)_{k+1}(w_2)$. Let

$$x_j = (\lim_{l>\max\{i,k\}}\; \epsilon^{\circ}_{i,l}(\epsilon_{k,l}\, w_j))_{i\geqslant 0}$$

for $j = 0, 1, 2$. Then, $x_2 \overset{k}{=} x_1 \circ x_0$ holds in W. From the non-expansiveness and monotonicity of g, it follows that

$$g(x_2) \overset{k+1}{=} g(x_1 \circ x_0) \supseteq g(x_1),$$

and, therefore, that

$$(g\, x_1)_{[k+1]} \subseteq (g\, x_2)_{[k+1]},$$

which yields the claimed monotonicity of $\imath(g)_{k+1}$. Finally, $\imath\, g \in W$ holds since the definition of $\imath$ satisfies $\epsilon^{\circ}_k(\imath\, g)_{k+1} = (\imath\, g)_k$ for all k.

Lemma 7. The assignment of g to $\imath(g)$ determines a non-expansive function from $\frac{1}{2}\cdot W \to_{mon} URel(Heap)$ to W, with a non-expansive inverse given by $\imath^{\bullet}$.

Proof (sketch). The non-expansiveness of $\imath$ and $\imath^{\bullet}$ is easy to see. To show that $\imath^{\bullet}$ is a right-inverse to $\imath$, we first prove that $(\imath(\imath^{\bullet}\, w))_n \overset{n}{=} w_n$ holds for all $w \in W$ and $n \in \mathbb{N}$. This yields the required equality, since

$$(\imath(\imath^{\bullet}\, w))_l \;=\; \lim_{n>l} \epsilon^{\circ}_{l,n}((\imath(\imath^{\bullet}\, w))_n) \;=\; \lim_{n>l} \epsilon^{\circ}_{l,n}(w_n) \;=\; w_l$$

follows for each l by the non-expansiveness of the $\epsilon^{\circ}_{n,l}$'s.

The fact that $\imath^{\bullet}$ is also a left-inverse to $\imath$ can be seen by a similar calculation. □

To complete the proof of Theorem 4, we need to establish the relationship between the monoid multiplication and the isomorphism.

Lemma 8. For all $w_1, w_2, w \in W$,

$$\iota^{-1}(w_1 \circ w_2)(w) \;=\; \iota^{-1}(w_1)(w_2 \circ w) \;*\; \iota^{-1}(w_2)(w).$$

Proof (sketch). We first show that $g(w) = \lim_k \lim_{j>k+1}(\iota\, g)_j(\epsilon^{\circ}_{k,j-1}\, w_k)$ for all g in $\frac{1}{2} \cdot W \to_{mon} URel(Heap)$ and $w \in W$. Using this equation, the claim is then established by unfolding the definitions of $\circ$ and ι^{-1}. □

6. Hereditarily monotone recursive worlds

In this section we present an alternative construction of a set of recursive worlds, which differs from the one defined in the previous section in several respects. Either set may be used for the interpretation of the capability calculus.

6.1. *Recursive worlds*

The first step in this construction is the definition of recursive worlds without a monotonicity condition. It is well known that one can solve recursive domain equations in **CBUlt**, given by locally contractive functors, by an adaptation of the inverse-limit method from classical domain theory (America and Rutten 1989). In particular, by considering the space of contractive but not necessarily monotone functions in the domain equation (18) above, America and Rutten's existence theorem applies.

Proposition 9. There exists a unique (up to isomorphism) metric space $(X, d) \in$ **CBUlt** and an isomorphism ι from $\frac{1}{2} \cdot X \to URel(Heap)$ to X.

Proof. X is obtained by America and Rutten's existence theorem for fixed points of locally contractive functors (America and Rutten 1989), applied to the functor F : $\mathbf{CBUlt}^{op} \longrightarrow$ **CBUlt**, defined by $F(X) = \frac{1}{2} \cdot X \to URel(Heap)$. □

The next step is to define the composition operation $\circ$ on X.

Lemma 10. There exists a non-expansive operation $\circ : X \times X \to X$ such that

$$\forall x_1, x_2, x \in X.\; \iota^{-1}(x_1 \circ x_2)(x) \;=\; \iota^{-1}(x_1)(x_2 \circ x) \;*\; \iota^{-1}(x_2)(x).$$

This operation is associative, and has $emp = \iota(I)$ as left and right unit for $I(w) = \mathbb{N} \times Heap$ the unit of the lifted separating conjunction described in Proposition 3.

Proof. The operation $\circ$ can be defined by a straightforward application of Banach's fixed-point theorem to the complete ultrametric space $X \times X \to X$. The proof that *emp* is a left and right unit is easy. For associativity, we prove

$$x_1 \circ (x_2 \circ x_3) \overset{n}{=} (x_1 \circ x_2) \circ x_3$$

for all $n \in \mathbb{N}$ by induction – see Schwinghammer *et al.* (2009). □

We define $f \otimes x$, for $f : \frac{1}{2} \cdot X \to URel(Heap)$ and $x \in X$, as the non-expansive function $\frac{1}{2} \cdot X \to URel(Heap)$ given by $(f \otimes x)(x') = f(x \circ x')$.

Since $\circ$ defines a monoid structure on X, there is an induced preorder on X given by

$$x \sqsubseteq y \;\Leftrightarrow\; \exists x_0.\ y = x \circ x_0.$$

We will now 'carve out' a subset of functions in $\frac{1}{2} \cdot X \to \mathit{URel}(\mathit{Heap})$ that are monotonic with respect to this preorder. This subset needs to be defined recursively.

6.2. *Relations on ultrametric spaces*

For $X \in \mathbf{CBUlt}$, let $\mathscr{R}(X)$ be the collection of all non-empty and closed[†] relations $R \subseteq X$; we will just write $\mathscr{R}$ when X is clear from the context. We set

$$R_{[n]} \overset{def}{=} \{y \mid \exists x \in X.\ x \overset{n}{=} y \ \wedge\ x \in R\}$$

for $R \in \mathscr{R}$. Thus, $R_{[n]}$ is the set of all points within distance 2^{-n} of R. Note that $R_{[n]} \in \mathscr{R}$. In fact, $\emptyset \neq R \subseteq R_{[n]}$ holds by the reflexivity of n-equality, and if $(y_k)_{k\in\mathbb{N}}$ is a sequence in $R_{[n]}$ with limit y in X, then $d(y_k, y) \leqslant 2^{-n}$ must hold for some k, that is, $y_k \overset{n}{=} y$. So there exists $x \in X$ with $x \in R$ and $x \overset{n}{=} y_k$, and hence, by transitivity, $x \overset{n}{=} y$, which then gives $\lim_n y_n \in R_{[n]}$.

We will now make some further observations that follow from the properties of n-equality on X. First, $R \subseteq S$ implies $R_{[n]} \subseteq S_{[n]}$ for any $R, S \in \mathscr{R}$. Moreover, using the fact that the n-equalities become increasingly fine, it follows that $(R_{[m]})_{[n]} = R_{[\min(m,n)]}$ for all $m, n \in \mathbb{N}$, so, in particular, each $(\cdot)_{[n]}$ is a closure operation on $\mathscr{R}$. As a consequence, we have

$$R \subseteq \ldots \subseteq R_{[n]} \subseteq \ldots \subseteq R_{[1]} \subseteq R_{[0]}.$$

By the 1-boundedness of X, we have $R_{[0]} = X$ for all $R \in \mathscr{R}$. Finally, $R = S$ if and only if $R_{[n]} = S_{[n]}$ for all $n \in \mathbb{N}$.

Proposition 11. Let $d : \mathscr{R} \times \mathscr{R} \to \mathbb{R}$ be defined by

$$d(R, S) = \inf\,\{2^{-n} \mid R_{[n]} = S_{[n]}\}.$$

Then $(\mathscr{R}, d)$ is a complete, 1-bounded, non-empty ultrametric space. The limit of a Cauchy chain $(R_n)_{n\in\mathbb{N}}$ with $d(R_n, R_{n+1}) \leqslant 2^{-n}$ is given by $\bigcap_n (R_n)_{[n]}$, and, in particular, $R = \bigcap_n R_{[n]}$ for any $R \in \mathscr{R}$.

Proof. First, $\mathscr{R}$ is non-empty since it contains X itself, and d is well defined since $R_{[0]} = S_{[0]}$ holds for any $R, S \in \mathscr{R}$. Next, since $R = S$ is equivalent to $R_{[n]} = S_{[n]}$ for all $n \in \mathbb{N}$, it follows that $d(R, S) = 0$ if and only if $R = S$. The fact that the ultrametric inequality $d(R, S) \leqslant \max\{d(R, T), d(T, S)\}$ holds is immediate by the definition of d, as is the fact that d is symmetric and 1-bounded.

To show completeness, we assume that $(R_n)_{n\in\mathbb{N}}$ is a Cauchy sequence in $\mathscr{R}$. Without loss of generality, we may also assume that $d(R_n, R_{n+1}) \leqslant 2^{-n}$ holds for all $n \in \mathbb{N}$, and,

[†] Recall that a relation is closed if it is closed under the limit operation.

therefore, that $(R_n)_{[n]} = (R_{n+1})_{[n]}$ for all $n \geqslant 0$. Writing S_n for $(R_n)_{[n]}$, we define $R \subseteq X$ by

$$R \stackrel{def}{=} \bigcap_{n \geqslant 0} S_n.$$

R is closed since each S_n is closed. We now prove that R is non-empty and, therefore, that $R \in \mathscr{R}$, by inductively constructing a sequence $(x_n)_{n \in \mathbb{N}}$ with $x_n \in S_n$. Let x_0 be an arbitrary element in $S_0 = X$. Having chosen $x_0, \ldots, x_n$, we pick some $x_{n+1} \in S_{n+1}$ such that $x_{n+1} \stackrel{n}{=} x_n$; this is always possible because $S_n = (S_{n+1})_{[n]}$ by our assumption on the sequence $(R_n)_{n \in \mathbb{N}}$. This is clearly a Cauchy sequence in X, and from $S_n \supseteq S_{n+1}$, it follows that $(x_n)_{n \geqslant k}$ is in fact a sequence in S_k for each $k \in \mathbb{N}$. But then $\lim_{n \in \mathbb{N}} x_n$ is also in S_k for each k, and thus also in R.

We now prove that R is the limit of the sequence $(R_n)_{n \in \mathbb{N}}$. By the definition of d it suffices to show that $R_{[k]} = (R_k)_{[k]}$ for all $k \geqslant 1$, or, equivalently, that $R_{[k]} = S_k$. From the definition of R, we have $R \subseteq S_k$, which immediately entails $R_{[k]} \subseteq (S_k)_{[k]} = S_k$.

To prove the other direction, that is, $S_k \subseteq R_{[k]}$, we assume that $x \in S_k$. To show that $x \in R_{[k]}$, we inductively construct a Cauchy sequence $(x_n)_{n \geqslant k}$ with $x_n \in S_n$, $x_k = x$ and $x_{n+1} \stackrel{n}{=} x_n$ analogously to the one above. Then $\lim_m x_m$ is in S_n for each $n \geqslant 0$, and thus also in R. Since $d_X(x_k, \lim_{n \geqslant k} x_n) \leqslant 2^{-k}$ by the ultrametric inequality, $x_k \in R_{[k]}$, or, equivalently, $x \in R_{[k]}$. □

6.3. *Hereditarily monotone recursive worlds*

We will now define the set of hereditarily monotonic functions W as a recursive predicate on the space X from Proposition 9. Let the function $\Phi : \mathscr{P}(X) \to \mathscr{P}(X)$ on subsets of X be given by

$$\Phi(R) = \{\iota(g) \mid \forall x, x_0 \in R.\ g(x) \subseteq g(x \circ x_0)\}.$$

The function restricts to a contractive function on $\mathscr{R}$ as follows.

Lemma 12. If $R \in \mathscr{R}$, then $\Phi(R)$ is non-empty and closed, and $R \stackrel{n}{=} S$ implies $\Phi(R) \stackrel{n+1}{=} \Phi(S)$.

Proof. It is clear that $\Phi(R) \neq \emptyset$ since $\iota(g) \in \Phi(R)$ for every constant function g from $\frac{1}{2} \cdot X$ to $URel(Heap)$. Limits of Cauchy chains in $\frac{1}{2} \cdot X \to URel(Heap)$ are given pointwise, so

$$(\lim_n g_n)(x) \subseteq (\lim_n g_n)(x \circ x_0)$$

holds for all Cauchy chains $(g_n)_{n \in \mathbb{N}}$ in $\Phi(R)$ and all $x, x_0 \in R$. This proves $\Phi(R) \in \mathscr{R}$.

We now show that Φ is contractive. To this end, we let $n \geqslant 0$ and assume $R \stackrel{n}{=} S$. Let $\iota(g) \in \Phi(R)_{[n+1]}$. We must show that $\iota(g) \in \Phi(S)_{[n+1]}$. By definition of the closure operation, there exists $\iota(f) \in \Phi(R)$ such that g and f are $(n{+}1)$-equal. Set $h(w) = f(w)_{[n+1]}$. Then h and g are also $(n{+}1)$-equal, so it suffices to show that $\iota(h) \in \Phi(S)$. To establish the latter, let $w_0, w_1 \in S$ be arbitrary. By the assumption that R and S are n-equal, there exist elements $w_0', w_1' \in R$ such that $w_0' \stackrel{n}{=} w_0$ and $w_1' \stackrel{n}{=} w_1$ holds in X, or, equivalently, such that w_0' and w_0 as well as w_1' and w_1 are $(n{+}1)$-equal in $\frac{1}{2} \cdot X$. By the non-expansiveness of $\circ$, this implies that $w_0' \circ w_1'$ and $w_0 \circ w_1$ are also $(n+1)$-equal in $\frac{1}{2} \cdot X$. Since

$$f(w_0) \overset{n+1}{=} f(w_0') \subseteq f(w_0' \circ w_1') \overset{n+1}{=} f(w_0 \circ w_1)$$

holds by the non-expansiveness of f and the assumption that $\imath(f) \in \Phi(R)$, we obtain the required inclusion $h(w_0) \subseteq h(w_0 \circ w_1)$ by the definition of h. $\square$

By Proposition 11 and the Banach theorem, we can now define the hereditarily monotonic functions W as the uniquely determined fixed point of Φ.

Theorem 13 (Existence of hereditarily monotone recursive worlds). There exists a non-empty and closed subset $W \subseteq X$ satisfying the condition

$$w \in W \iff \exists g.\ w = \imath(g) \ \wedge\ \forall w_1, w_2 \in W.\ g(w_1) \subseteq g(w_1 \circ w_2).$$

Note that W constructed in this way does not quite satisfy the conditions stated in Theorem 4: we do not have an isomorphism between W and the non-expansive and monotonic functions from W (viewed as an ultrametric space itself), but rather between W and all functions from X that *restrict* to monotonic functions whenever applied to hereditarily monotonic arguments. Bearing this in mind, we abuse notation and write

$$\tfrac{1}{2} \cdot W \to_{mon} URel(A) = \{g : \tfrac{1}{2} \cdot X \to URel(A) \mid \forall w_1, w_2 \in W.\ g(w_1) \subseteq g(w_1 \circ w_2)\}.$$

Then, for our particular application of interest, we also have to ensure that all the operations restrict appropriately (*cf.* Section 7). Here, as a first step, we will show that the composition operation $\circ$ restricts to W.

Lemma 14. For all $n \in \mathbb{N}$, if $w_1, w_2 \in W$, then $w_1 \circ w_2 \in W_{[n]}$. In particular, since $W = \bigcap_n W_{[n]}$, it follows that $w_1, w_2 \in W$ implies $w_1 \circ w_2 \in W$.

Proof. The proof is by induction on n.

The base case is immediate since $W_{[0]} = X$.

So we suppose $n > 0$ and let $w_1, w_2 \in W$. We must prove that $w_1 \circ w_2 \in W_{[n]}$. Let w_1' be such that $\imath^{-1}(w_1')(w) = \imath^{-1}(w_1)(w)_{[n]}$. Observe that $w_1' \in W$, that w_1' and w_1 are n-equal, and that w_1' is such that n-equality of w, w' in $\frac{1}{2} \cdot X$ already implies

$$\imath^{-1}(w_1')(w) = \imath^{-1}(w_1')(w').$$

Since w_1' and w_1 are n-equivalent, the non-expansiveness of the composition operation implies

$$w_1 \circ w_2 \overset{n}{=} w_1' \circ w_2.$$

Thus, it suffices to show that $w_1' \circ w_2 \in W = \Phi(W)$. To see this, let $w, w_0 \in W$ be arbitrary, and note that by the induction hypothesis we have $w_2 \circ w \in W_{[n-1]}$. This means that there exists $w' \in W$ such that $w' \overset{n}{=} w_2 \circ w$ holds in $\frac{1}{2} \cdot X$. Hence

$$\begin{aligned}
\iota^{-1}(w_1' \circ w_2)(w) &= \iota^{-1}(w_1')(w_2 \circ w) * \iota^{-1}(w_2)(w) && \text{(by the definition of } \circ)\\
&= \iota^{-1}(w_1')(w') * \iota^{-1}(w_2)(w) && \text{(by } w' \overset{n}{=} w_2 \circ w)\\
&\subseteq \iota^{-1}(w_1')(w' \circ w_0) * \iota^{-1}(w_2)(w \circ w_0) && \text{(by hereditariness)}\\
&= \iota^{-1}(w_1')((w_2 \circ w) \circ w_0) * \iota^{-1}(w_2)(w \circ w_0) && \text{(by } w' \overset{n}{=} w_2 \circ w)\\
&= \iota^{-1}(w_1' \circ w_2)(w \circ w_0) && \text{(by the definition of } \circ)
\end{aligned}$$

Since w and w_0 were chosen arbitrarily, this calculation establishes $w_1' \circ w_2 \in W$. □

Moreover, the BI algebra structure that exists on $\frac{1}{2} \cdot X \to URel(Heap)$ by Proposition 3 restricts to the hereditarily monotone functions.

Proposition 15. $\frac{1}{2} \cdot W \to_{mon} URel(Heap)$ forms a complete BI algebra where the operations are non-expansive. Meets and joins are given by the pointwise extension of intersection and union on $URel(Heap)$, and $f \Rightarrow g$ is defined by

$$(f \Rightarrow g)(x) = \bigcap_{x_0 \in X} \big(f(x \circ x_0) \Rightarrow g(x \circ x_0)\big).$$

The separating conjunction $f * g$ and its unit I are defined pointwise, and the separating implication $f \mathrel{-\!\!*} g$ is defined by

$$(f \mathrel{-\!\!*} g)(x) = \bigcap_{x_0 \in X} \big(f(x \circ x_0) \mathrel{-\!\!*} g(x \circ x_0)\big)$$

from the separating implication on $URel(Heap)$.

7. Step-indexed possible world semantics of capabilities

In this section we prove the soundness of the calculus of capabilities. After defining the semantic domains for the interpretation of types and capabilities, we give the full syntax and typing rules for the system presented in Section 2. Then, using the hereditarily monotone recursive worlds W, we construct a model of types and capabilities based on the operational semantics.

Alternatively, we could use the monotone recursive worlds from Section 5 instead, and this would require only minor and straightforward modifications of the interpretation below.

7.1. *Semantic domains and constructors*

Let $X \in \mathbf{CBUlt}$ denote the solution to the ultrametric equation

$$X \cong \tfrac{1}{2} \cdot X \to URel(Heap)$$

from Proposition 9, and let $W \in \mathscr{R}(X)$ denote the subset of hereditarily monotone recursive worlds (Theorem 13).

We define semantic domains for the capabilities and the value and memory types:

$$\begin{aligned} Cap &= \tfrac{1}{2}\cdot W \to_{mon} URel(Heap) \\ VT &= \tfrac{1}{2}\cdot W \to_{mon} URel(Val) \\ MT &= \tfrac{1}{2}\cdot W \to_{mon} URel(Val \times Heap) \end{aligned}$$

such that $g \in Cap$ if and only if $\iota(g) \in W$.

To define operations on the semantic domains that correspond to the syntactic type and capability constructors, we consider the lifting of (memory) types from values to expressions.

Definition 16 (Expression typing). Consider $f : \frac{1}{2}\cdot X \to URel(Val \times Heap)$. The function $\mathscr{E}(f) : X \to URel(Exp \times Heap)$ is defined by $(k,(t,h)) \in \mathscr{E}(f)(x)$ if and only if

$$\forall j \leqslant k, t', h'.\ (t \,|\, h) \longmapsto^{j} (t' \,|\, h') \ \wedge\ (t' \,|\, h') \text{ irreducible}$$
$$\Rightarrow\ (k{-}j,(t',h')) \in \bigcup_{w\in W} f(x\circ w) * \iota^{-1}(x\circ w)(emp).$$

Note that it is at this point that the indexing by natural numbers that allows one to measure the distance between uniform relations is linked to the operational semantics of the programming language.

Also note that in this definition, f is a contractive function on X whereas $\mathscr{E}(f)$ is merely non-expansive. This is because the conclusion uses the world x as a heap predicate, qua $\iota^{-1}(x\circ w)(emp)$, that is, the scaling by $1/2$ is undone, and the number of steps j taken in the reduction sequence may in fact be 0.

Lemma 17. Let $f : \frac{1}{2}\cdot X \to URel(Val \times Heap)$. Then $\mathscr{E}(f)$ is non-expansive, and for all $x \in X$, we have $\mathscr{E}(f)(x) \in URel(Exp \times Heap)$. Moreover, the assignment of $\mathscr{E}(f)$ to f is non-expansive.

Proof. Observe that $f \overset{n}{=} f'$ and $x \overset{n}{=} x'$ in X implies

$$\begin{gathered} f(x\circ w) \overset{n}{=} f'(x'\circ w) \\ \iota^{-1}(x\circ w)(emp) \overset{n}{=} \iota^{-1}(x'\circ w)(emp) \end{gathered}$$

for any $w \in W$, by the non-expansiveness of f, f' and $\circ$. Thus

$$\mathscr{E}(f)(x) \overset{n}{=} \mathscr{E}(f')(x').$$

In particular, for $f = f'$ we obtain the non-expansiveness of $\mathscr{E}(f)$, and for $x = x'$ we obtain the non-expansiveness of $\mathscr{E}$ by definition of the sup metric. $\square$

Definition 18 (Capability and type constructors). In addition to separating conjunction and its unit, given in Proposition 15, we define the following operations:

Invariant extension:

Let $g : \frac{1}{2} \cdot X \to URel(A)$ and $w \in W$. We define

$$g \otimes w : \tfrac{1}{2} \cdot X \to URel(A)$$

by

$$(g \otimes w)(x) = g(w \circ x).$$

Separation:

Let $p \in URel(A \times Heap)$ and $r \in URel(Heap)$. We define

$$p * r \in URel(A \times Heap)$$

by

$$p * r = \{(k, (a, h \cdot h')) \mid (k, (a, h)) \in p \ \wedge \ (k, h') \in r\}.$$

This operation can be lifted pointwise:

$$(g * c)(x) = g(x) * c(x)$$

for

$$\begin{aligned} g &: \tfrac{1}{2} \cdot X \to URel(A \times Heap) \\ c &: \tfrac{1}{2} \cdot X \to URel(Heap). \end{aligned}$$

For notational convenience, we will sometimes view $r \in URel(Heap)$ as the constant function that maps any $x \in X$ to r, and thus write $g * r$ for this pointwise lifting.

Singleton capabilities:

Let $v \in Val$ and $g : \frac{1}{2} \cdot X \to URel(Val \times Heap)$. We define

$$\{v : g\} : \tfrac{1}{2} \cdot X \to URel(Heap)$$

by

$$\{v : g\}(x) = \{(k, h) \mid (k, (v, h)) \in g(x)\}.$$

Name abstraction:

Let $F : Val \to (\frac{1}{2} \cdot X \to URel(A))$, where Val is viewed as a discrete ultrametric space. We define

$$\exists F : \tfrac{1}{2} \cdot X \to URel(A)$$

by

$$(\exists F)(x) = \bigcup_{v \in Val} F(v)(x).$$

Universal quantification:

Let S be a set (viewed as an object of **CBUlt** with discrete metric), and let $F : S \to (\frac{1}{2} \cdot X \to URel(A))$. We define $\forall F : \frac{1}{2} \cdot X \to URel(A)$ by

$$(\forall F)(x) = \bigcap_{s \in S} F(s)(x).$$

Recursion:

Let $F : (\frac{1}{2} \cdot X \to URel(A)) \to (\frac{1}{2} \cdot X \to URel(A))$ be a contractive function. We define

$$\mathit{fix}\, F : \tfrac{1}{2} \cdot X \to URel(A)$$

by

$$\mathit{fix}\, F = \text{ the unique } g : \tfrac{1}{2} \cdot X \to URel(A) \text{ such that } g = F(g),$$

which exists by the Banach fixed-point theorem.

Sum types:

Let $g_1, g_2 : \frac{1}{2} \cdot X \to URel(Val)$. We define

$$g_1 + g_2 : \tfrac{1}{2} \cdot X \to URel(Val)$$

by

$$(g_1 + g_2)(x) = \{(k, \mathsf{inj}^i v) \mid \forall j < k.\ (j, v) \in g_i(x)\}.$$

Similarly, for $g_1, g_2 : \frac{1}{2} \cdot X \to URel(Val \times Heap)$, we define

$$g_1 + g_2 : \tfrac{1}{2} \cdot X \to URel(Val \times Heap)$$

by

$$(g_1 + g_2)(x) = \{(k, (\mathsf{inj}^i v, h)) \mid \forall j < k.\ (j, (v, h)) \in g_i(x)\}.$$

Product types:

Let $g_1, g_2 : \frac{1}{2} \cdot X \to URel(Val)$. We define

$$g_1 \times g_2 : \tfrac{1}{2} \cdot X \to URel(Val)$$

by

$$(g_1 \times g_2)(x) = \{(k, \langle v_1, v_2 \rangle) \mid \forall j < k.\ (j, v_1) \in g_1(x) \wedge (j, v_2) \in g_2(x)\}.$$

Similarly, for $g_1, g_2 : \frac{1}{2} \cdot X \to URel(Val \times Heap)$ we define

$$g_1 \times g_2 : \tfrac{1}{2} \cdot X \to URel(Val \times Heap)$$

by

$$(g_1 \times g_2)(x) = \{(k, (\langle v_1, v_2 \rangle, h_1 \cdot h_2)) \mid \forall j < k.\ (j, (v_i, h_i)) \in g_i(x)\}.$$

Arrow types:

Let $g_1, g_2 : \frac{1}{2} \cdot X \to URel(Val \times Heap)$. We define

$$g_1 \to g_2 : \tfrac{1}{2} \cdot X \to URel(Val)$$

on $x \in X$ by

$$\begin{aligned}\{(k, \mathsf{fun}\, f(y) = t) \mid \forall j < k.\ \forall w \in W.\ \forall r \in URel(Heap). \forall v, h.&\\ (j, (v, h)) \in g_1(x \circ w) * \iota^{-1}(x \circ w)(emp) * r \ \Rightarrow&\\ (j, (t[f{:=}\mathsf{fun}\, f(y) = t, y{:=}v], h)) \in \mathscr{E}(g_2 * r)(x \circ w)\}.&\end{aligned}$$

Reference types:

Let $g : \frac{1}{2} \cdot X \to URel(Val \times Heap)$. We define

$$\mathit{ref}(g) : \tfrac{1}{2} \cdot X \to URel(Val)$$

by

$$\mathit{ref}(g)(x) \ = \{(k,(l,h\cdot[l\mapsto v]))\mid \forall j<k.\ (j,(v,h))\in g(x)\}.$$

The case for arrow types realises the key ideas of our model that we described in Section 3 as follows. First, the universal quantification over $w \in W$ and subsequent use of the world $x \circ w$ builds in monotonicity, and, intuitively, means that $g_1 \to g_2$ is parametric in (and hence preserves) invariants that have been added by the procedure's context. In particular, the definition states that procedure application preserves this invariant when viewed as the predicate $\iota^{-1}(x\circ w)(\mathit{emp})$. By also conjoining r as an invariant, we 'bake in' the first-order frame property, which results in a subtyping axiom $\chi_1 \to \chi_2 \leqslant \chi_1 * C \to \chi_2 * C$ in the type system. The existential quantification over w' in the definition of $\mathscr{E}$ allows us to 'absorb' a part of the local heap description into the world. Finally, the quantification over indices $j < k$ in the definition of $g_1 \to g_2$ ensures that $(g_1 \to g_2)(x)$ is uniform. There are three reasons we require j to be *strictly* less than k. Technically, as for the definition of $\mathscr{E}$, the use of $\iota^{-1}(x \circ w)$ in the definition undoes the scaling by $1/2$, and $j < k$ ensures the non-expansiveness of $g_1 \to g_2$ as a function $1/2 \cdot X \to \mathit{URel}(\mathit{Val})$. Moreover, it lets us prove the typing rule for *recursive* functions by induction on k. Finally, it means that $\to$ is a contractive type constructor, which justifies the formal contractiveness assumption about arrow types that we made earlier. Intuitively, the use of $j < k$ for the arguments suffices since application consumes a step. The use of $j < k$ in sum, product and reference types instead of $j \leqslant k$ ensures that these constructors are contractive in their arguments and not merely non-expansive.

The definition of $\mathit{ref}(g)$ builds in separation. If the semantic memory type g describes a value v together with a heap fragment h, then the semantic memory type $\mathit{ref}(g)$ describes a memory location l together with the heap fragment $h \cdot [l \mapsto v]$. By definition, the combination $h \cdot [l \mapsto v]$ exists only if l is not in the domain of h. The definition of $g_1 \times g_2$, in the case of value/heap pairs, also builds in separation. In Section 7.2, we will use these definitions to interpret syntactic memory types, so (for instance) the syntactic memory type $\mathsf{ref}\,(\mathsf{ref}\,\mathsf{int} \times \mathsf{ref}\,\mathsf{int})$ describes a configuration that must consist of three distinct memory cells. In short, taken together, the memory type constructors $\times$, $+$, ref and μ describe the (mutable) algebraic data structures *without sharing or cycles*.

This does not mean that the type system or the semantic model are unable to describe cyclic structures in the heap. A memory cell that contains its own address, for instance, can be described by the memory type $\exists\sigma.([\sigma] * \{\sigma : \mathsf{ref}\,[\sigma]\})$. (It is *not* described by the recursive memory type $\mu\beta.\mathsf{ref}\,\beta$, which describes an infinite chain of pairwise distinct memory cells, and is therefore empty!) As illustrated by this example, singleton regions are a mechanism for describing situations where there is *sharing* (they are inspired by earlier work on alias types (Smith *et al.* 2000)). What the present type system lacks is a mechanism for describing situations where there is *aliasing*, that is, situations where it is not statically known exactly whether two memory addresses are equal or distinct. As a simple example of such a situation, think of a circular list of references, where the length of the list is not statically known. There are several possibilities for describing such a data structure: for instance, one could extend the system with Charguéraud and Pottier's group regions; or one could extend it with untagged unions of the form $\theta_1 \vee \theta_2$, and use these untagged unions to define untagged list segments in the style of separation logic (Reynolds 2002), and then use a memory type of the form $\exists\sigma.([\sigma] * \{\sigma : \mathsf{listseg}\,\sigma\})$ to describe a circular list of unknown length. The type system presented here offers neither of these features. However, our semantic model should be able to support them without difficulty.

Lemma 19 (Well-definedness). The operations given in Definition 18 are well defined, that is, each operation is a non-expansive function that maps into uniform relations of the right kind. Moreover, they restrict to non-expansive operations on monotonic functions:

— If

$$g : \tfrac{1}{2} \cdot W \to_{mon} URel(A),$$

then

$$g \otimes w : \tfrac{1}{2} \cdot W \to_{mon} URel(A).$$

The operation $g, w \mapsto g \otimes w$ is non-expansive in g and contractive in w.

— If

$$g : \tfrac{1}{2} \cdot W \to_{mon} URel(A \times Heap)$$

and $c \in Cap$, then

$$g * c : \tfrac{1}{2} \cdot W \to_{mon} URel(A \times Heap).$$

The operation $g, c \mapsto g * c$ is non-expansive in g and c.

— If

$$g \in MT,$$

then

$$\{v : g\} \in Cap.$$

The operation $g \mapsto \{v : g\}$ is non-expansive.

— If

$$F : Val \to (\tfrac{1}{2} \cdot W \to_{mon} URel(A)),$$

then

$$\exists F : \tfrac{1}{2} \cdot W \to_{mon} URel(A).$$

The operation $F \mapsto \exists F$ is non-expansive.

— If

$$F : S \to (\tfrac{1}{2} \cdot W \to_{mon} URel(A)),$$

then

$$\forall F : \tfrac{1}{2} \cdot W \to_{mon} URel(A).$$

The operation $F \mapsto \forall F$ is non-expansive.

— If

$$F : (\tfrac{1}{2} \cdot W \to_{mon} URel(A)) \to (\tfrac{1}{2} \cdot W \to_{mon} URel(A))$$

is contractive, then

$$\mathit{fix}\, F : \tfrac{1}{2} \cdot W \to_{mon} URel(A).$$

The operation $F \mapsto \mathit{fix}\, F$ is non-expansive.

— If

$$g_1, g_2 \in VT,$$

then

$$g_1 + g_2 \in VT,$$

and if

$$g_1, g_2 \in MT,$$

then

$$g_1 + g_2 \in MT.$$

The operations $g_1, g_2 \mapsto g_1 + g_2$ are contractive in g_1 and g_2.

— If

$$g_1, g_2 \in VT,$$

then

$$g_1 \times g_2 \in VT,$$

and if

$$g_1, g_2 \in MT,$$

then

$$g_1 \times g_2 \in MT.$$

The operations $g_1, g_2 \mapsto g_1 \times g_2$ are contractive in g_1 and g_2.

— If

$$g_1, g_2 \in MT,$$

then

$$g_1 \to g_2 \in VT.$$

The operation $g_1, g_2 \mapsto g_1 \to g_2$ is contractive in g_1 and g_2.

— If

$$g \in MT,$$

then

$$\mathit{ref}(g) \in MT.$$

The operation $g \mapsto \mathit{ref}(g)$ is contractive.

Proof. We will just consider the cases of invariant extension and sum types in detail:

— Case $g : \frac{1}{2} \cdot X \to URel(A)$ and $w \in W$:

By definition,

$$(g \otimes w)(x) = g(w \circ x)$$

is a uniform relation on A for any $x \in X$. By the non-expansiveness of g and $\circ$ (*cf.* Lemma 10), $g \otimes w$ is a non-expansive function.

Next we show that $\otimes$ restricts to the monotone functions. Assume

$$g : \tfrac{1}{2} \cdot W \to_{mon} URel(A).$$

To show

$$g \otimes w : \tfrac{1}{2} \cdot W \to_{mon} URel(A),$$

we must prove

$$(g \otimes w)(w_1) \subseteq (g \otimes w)(w_1 \circ w_2)$$

for all $w_1, w_2 \in W$. Note that $w \in W$, and thus Lemma 14 shows $w \circ w_1 \in W$. Hence,

$$g(w \circ w_1) \subseteq g(w \circ w_1 \circ w_2)$$

by the assumption

$$g : \tfrac{1}{2} \cdot W \to_{mon} URel(A),$$

and the claim then follows from the definition of $g \otimes w$.
We now show that $\otimes$ is non-expansive in its first and contractive in its second argument. If $g \overset{n}{=} g'$, then

$$(g \otimes w)(x) \overset{n}{=} (g' \otimes w)(x)$$

by the definition of the sup-metric, which means that $g \mapsto g \otimes w$ is non-expansive. Finally, assuming we have $w \overset{n}{=} w'$ for $w, w' \in W$, we get

$$w \circ x \overset{n+1}{=} w' \circ x$$

holds in $\frac{1}{2} \cdot X$ for any $x \in X$ by the non-expansiveness of $\circ$ and the scaling operation. Thus

$$(g \otimes w)(x) \overset{n+1}{=} (g \otimes w')(x)$$

follows from the non-expansiveness of g, and since x was chosen arbitrarily the definition of the sup-metric yields

$$g \otimes w \overset{n+1}{=} g \otimes w',$$

which shows that $w \mapsto g \otimes w$ is contractive.

— Case $g_1, g_1', g_2, g_2' : \frac{1}{2} \cdot X \to URel(Val)$, assuming $g_1 \overset{n}{=} g_1'$ and $g_2 \overset{n}{=} g_2'$:
For any $x, x' \in X$ such that $x \overset{n}{=} x'$ holds with respect to the metric on $\frac{1}{2} \cdot X$, the non-expansiveness of g_1, g_2 yields

$$g_1(x) \overset{n}{=} g_1(x')$$
$$g_2(x) \overset{n}{=} g_2(x').$$

Hence, $(j, v) \in g_1(x)$ if and only if $(j, v) \in g_1'(x')$ for any $j < n$, and $(j, v) \in g_2'(x)$ if and only if $(j, v) \in g_2'(x')$ for any $j < n$. By the definitions of $g_1 + g_2$ and $g_1' + g_2'$, it follows that

$$(g_1 + g_2)(x)_{[n+1]} = (g_1' + g_2')(x')_{[n+1]},$$

that is, that

$$(g_1 + g_2)(x) \overset{n+1}{=} (g_1 + g_2)(x').$$

From this observation, taking $g_1 = g_1'$ and $g_2 = g_2'$, it follows immediately that $g_1 + g_2$ is non-expansive. Moreover, taking $x = x'$, the definition of the sup-metric shows that the assignment $g_1, g_2 \mapsto g_1 + g_2$ is contractive.

Since $g_1(x)$ and $g_2(x)$ are uniform relations, it is easy to see that $(k, v) \in (g_1 + g_2)(x)$ implies $(j, v) \in (g_1 + g_2)(x)$ for all $j \leqslant k$. Finally, from the definition of $g_1 + g_2$, it follows that $g_1, g_2 \in VT$ implies $g_1 + g_2 \in VT$.

The remaining cases are similar. $\square$

7.2. *Type system and soundness*

The syntax and typing rules of Charguéraud and Pottier's capability type system are given in Figures 7, 8 and 9. In addition to the typing rules given earlier, the capability type system also features subtype and subcapability relations. Figure 10 shows some of the axioms that induce these relations. Axioms 23 and 24 allow the elimination of a universal quantifier and introduction of an existential quantifier. (We write $[\xi := \ldots]$ for a substitution that replaces the variable ξ with an object of the appropriate syntactic category, depending on the nature of ξ.) Axiom (25) is a variant of the first-order (shallow) frame rule from Figure 6[†]. Axiom (26) allows us to 'garbage-collect' capabilities for parts of the heap that are no longer needed. This axiom only holds in a 'non-tight' interpretation of assertions like the one we use here. Axioms (27) and (28) permit the translation back and forth between a value type τ and a singleton type $[\sigma]$ (together with a capability for σ). Many more axioms could be listed, which (we believe) can be proved to be sound with respect to the model and may even be useful in practice. Attempting to provide an exhaustive list would be a somewhat tedious exercise – see Charguéraud and Pottier (2008) and Pottier (2011), where many axioms are given.

The relation $\leqslant$ is defined inductively by inference rules (not shown here) that state that all type and capability constructors are covariant[‡], with two exceptions: as usual, arrow types are contravariant in their first argument, and $\otimes$ is invariant in its second argument.

The rules that define the typing judgement for values ($\Delta \vdash v : \tau$) include a rule for introducing a universal quantifier, whereas the rules that define the typing judgement

[†] The deep variant of this axiom, $\chi_1 \to \chi_2 \leqslant (\chi_1 \circ C) \to (\chi_2 \circ C)$, is not sound in the capability calculus. Pottier (2009b) gives a counterexample based on this axiom and the anti-frame rule, and a similar counterexample that does not use the anti-frame rule can be constructed along the lines of Schwinghammer *et al.* (2009, Proposition 1).

[‡] The references in the system of Charguéraud and Pottier (2008) are *strong references*. The type system regards reference types as affine memory types, not value types, and the system keeps explicit track of the ownership of reference cells. For this reason, the type system allows strong (type-changing) updates, and, for the same reason, it allows reference types to be considered covariant. (If θ_1 is a subtype of θ_2, then changing the type of a reference cell from $\mathsf{ref}\,\theta_1$ to $\mathsf{ref}\,\theta_2$ can be viewed as a particular case of a strong update, where no write instruction is required at runtime because the content of the cell does not change.) The references in ML's type system, on the other hand, are *weak references*. They do not have an explicit owner: they are ordinary values, and can be duplicated. For this reason, they cannot admit strong updates, and must be invariant. In Pottier's encoding of weak references in terms of strong references and the anti-frame rule (Pottier 2008), the invariance requirement arises from the use of the anti-frame rule, which requires that the hidden state be described by an *invariant*.

Variables	$\xi ::= \alpha \mid \beta \mid \gamma \mid \sigma$
Capabilities	$C ::= C \otimes C \mid \emptyset \mid C * C \mid \{\sigma : \theta\} \mid \exists\sigma.C \mid \gamma \mid \mu\gamma.C \mid \forall\xi.C$
Value types	$\tau ::= \tau \otimes C \mid 0 \mid 1 \mid \mathsf{int} \mid \tau + \tau \mid \tau \times \tau \mid \chi \to \chi \mid [\sigma] \mid \alpha \mid \mu\alpha.\tau \mid \forall\xi.\tau$
Memory types	$\theta ::= \theta \otimes C \mid \tau \mid \theta + \theta \mid \theta \times \theta \mid \mathsf{ref}\,\theta \mid \theta * C \mid \exists\sigma.\theta \mid \beta \mid \mu\beta.\theta \mid \forall\xi.\theta$
Computation types	$\chi ::= \chi \otimes C \mid \tau \mid \chi * C \mid \exists\sigma.\chi$
Value contexts	$\Delta ::= \Delta \otimes C \mid \varnothing \mid \Delta, x{:}\tau$
Affine contexts	$\Gamma ::= \Gamma \otimes C \mid \varnothing \mid \Gamma, x{:}\chi \mid \Gamma * C$

Fig. 7. Syntax of capabilities and types

$$\frac{(x:\tau)\in\Delta}{\Delta\vdash x:\tau} \qquad \frac{}{\Delta\vdash\langle\rangle:1} \qquad \frac{\Delta\vdash v:\tau_i}{\Delta\vdash(\mathsf{inj}^i\ v):(\tau_1+\tau_2)} \qquad \frac{\Delta\vdash v_1:\tau_1 \qquad \Delta\vdash v_2:\tau_2}{\Delta\vdash\langle v_1,v_2\rangle:(\tau_1\times\tau_2)}$$

$$\frac{\Delta, f:\chi_1\to\chi_2, x:\chi_1 \Vdash t:\chi_2}{\Delta\vdash\mathsf{fun}\,f(x)=t:\chi_1\to\chi_2} \qquad \begin{array}{c}\forall\text{-INTRO}\\ \dfrac{\begin{array}{c}\Delta\vdash v:\tau\\ \xi\notin\Delta\end{array}}{\Delta\vdash v:\forall\xi.\tau}\end{array} \qquad \begin{array}{c}\text{DEEP FRAME (VAL)}\\ \dfrac{\Delta\vdash v:\tau}{\Delta\otimes C\vdash v:\tau\otimes C}\end{array} \qquad \begin{array}{c}\text{SUB (VAL)}\\ \dfrac{\begin{array}{c}\Delta\vdash v:\tau'\\ \tau'\leqslant\tau\end{array}}{\Delta\vdash v:\tau}\end{array}$$

Fig. 8. Typing rules for values

for terms ($\Gamma \Vdash t : \chi$) do not include such a rule. This is *value restriction* (Wright 1995). Charguéraud and Pottier's system, without the anti-frame rule, does not require this restriction (Charguéraud and Pottier 2008), but once the anti-frame rule is introduced, the restriction becomes necessary for soundness, as noted and proved in Pottier (2009b; 2011). It is in a sense obvious why this restriction is required: the anti-frame rule allows encoding weak references, and it is well known that the combination of weak references and unrestricted polymorphism is unsound (Wright 1995). For a more technical explanation of why the restriction is necessary, one can check that our model does not validate a universal quantifier introduction rule for terms (because it would require commuting the existential quantifier over w in Definition 16 of semantic expression typing and the universal quantifier in Definition 18 of semantic universal quantification).

Using the operations given in Definition 18, the interpretation of capabilities and types is defined in Figure 11 by induction on the syntax. The interpretation depends on an environment η, which maps region names $\sigma \in \mathit{RegName}$ to closed values $\eta(\sigma) \in \mathit{Val}$, capability variables γ to semantic capabilities $\eta(\gamma) \in \mathit{Cap}$, and type variables α and β to semantic types $\eta(\alpha) \in \mathit{VT}$ and $\eta(\beta) \in \mathit{MT}$. By Lemma 19, we obtain interpretations

$$[\![C]\!]_\eta \in \mathit{Cap}$$
$$[\![\tau]\!]_\eta \in \mathit{VT}$$
$$[\![\theta]\!]_\eta \in \mathit{MT}.$$

$$\frac{\Delta \vdash v : \tau}{\Delta \Vdash v : \tau} \qquad \frac{\Delta \vdash v : \chi_1 \to \chi_2 \qquad \Delta, \Gamma \Vdash t : \chi_1}{\Delta, \Gamma \Vdash (v\, t) : \chi_2} \qquad \frac{\Gamma \Vdash v : \tau_1 \times \tau_2}{\Gamma \Vdash \mathsf{proj}^i\, v : \tau_i}$$

$$\frac{\begin{array}{c}\Delta \vdash v_1 : (\exists \sigma_1.[\sigma_1] * \{\sigma : [\sigma_1] + 0\} * \{\sigma_1 : \theta_1\} * C) \to \chi \\ \Delta \vdash v_2 : (\exists \sigma_2.[\sigma_2] * \{\sigma : 0 + [\sigma_2]\} * \{\sigma_2 : \theta_2\} * C) \to \chi \\ \Delta, \Gamma \Vdash v : [\sigma] * \{\sigma : \theta_1 + \theta_2\} * C\end{array}}{\Delta, \Gamma \Vdash \mathsf{case}(v_1, v_2, v) : \chi} \qquad \frac{\Gamma \Vdash v : \tau}{\Gamma \Vdash \mathsf{ref}\, v : \exists \sigma.[\sigma] * \{\sigma : \mathsf{ref}\, \tau\}}$$

$$\frac{\Gamma \Vdash v : [\sigma] * \{\sigma : \mathsf{ref}\, \tau\}}{\Gamma \Vdash \mathsf{get}\, v : \tau * \{\sigma : \mathsf{ref}\, \tau\}} \qquad \frac{\Gamma \Vdash v : ([\sigma] \times \tau_2) * \{\sigma : \mathsf{ref}\, \tau_1\}}{\Gamma \Vdash \mathsf{set}\, v : 1 * \{\sigma : \mathsf{ref}\, \tau_2\}} \qquad \frac{\Gamma, x : \chi_1 \Vdash t : \chi_2 \qquad \sigma \notin \Gamma, \chi_2}{\Gamma, x : \exists \sigma.\chi_1 \Vdash t : \chi_2}\ \textsc{$\exists$-Elim (comp)}$$

$$\frac{\Gamma * C \Vdash t : \chi_2 \qquad \sigma \notin \Gamma, \chi_2}{\Gamma * (\exists \sigma.C) \Vdash t : \chi_2}\ \textsc{$\exists$-Elim (cap)} \qquad \frac{\Gamma \Vdash t : \chi}{\Gamma * C \Vdash t : \chi * C}\ \textsc{Shallow frame} \qquad \frac{\Gamma \Vdash t : \chi}{(\Gamma \otimes C) * C \Vdash t : (\chi \otimes C) * C}\ \textsc{Deep frame (comp)}$$

$$\frac{\Gamma \otimes C \Vdash t : (\chi \otimes C) * C}{\Gamma \Vdash t : \chi}\ \textsc{Anti-frame} \qquad \frac{\Gamma \leqslant \Gamma' \qquad \Gamma' \Vdash t : \chi' \qquad \chi' \leqslant \chi}{\Gamma \Vdash t : \chi}\ \textsc{Sub (comp)}$$

Fig. 9. Typing rules for expressions

$$\begin{aligned}
\forall \xi.\tau &\leqslant \tau[\xi := \ldots] && (23)\\
\tau[\xi := \ldots] &\leqslant \exists \xi.\tau && (24)\\
\chi_1 \to \chi_2 &\leqslant (\chi_1 * C) \to (\chi_2 * C) && (25)\\
C &\leqslant \emptyset && (26)\\
\tau &\leqslant \exists \sigma.[\sigma] * \{\sigma : \tau\} && (27)\\
[\sigma] * \{\sigma : \tau\} &\leqslant \tau * \{\sigma : \tau\} && (28)
\end{aligned}$$

Fig. 10. Some subtyping axioms

Moreover, Lemma 19 shows that whenever C is formally contractive in ξ, then $g \mapsto [\![C]\!]_{\eta[\xi:=g]}$ is contractive (and similarly for formally contractive types τ and χ), which guarantees that the fixed points in Figure 11 are well defined.

The structural equivalences given in Figure 5 can be verified with respect to this interpretation. The monoid equations follow since $*$ and $\circ$ define monoid structures on *Cap*; the latter via the bijection ι between W and *Cap*. We will just consider the case of associativity of $\circ$.

Lemma 20. For all C_1, C_2, C_3, we have

$$[\![C_1 \circ (C_2 \circ C_3)]\!] = [\![(C_1 \circ C_2) \circ C_3]\!] .$$

Capabilities, $[\![C]\!]_\eta : 1/2 \cdot W \to_{mon} URel(Heap)$

$$
\begin{aligned}
[\![C_1 \otimes C_2]\!]_\eta &= [\![C_1]\!]_\eta \otimes \iota([\![C_2]\!]_\eta) & [\![\emptyset]\!]_\eta &= I \\
[\![C_1 * C_2]\!]_\eta &= [\![C_1]\!]_\eta * [\![C_2]\!]_\eta & [\![\{\sigma : \theta\}]\!]_\eta &= \{\eta(\sigma) : [\![\theta]\!]_\eta\} \\
[\![\gamma]\!]_\eta &= \eta(\gamma) & [\![\exists\sigma.C]\!]_\eta &= \exists(\lambda v \in Val.\ [\![C]\!]_{\eta[\sigma:=v]}) \\
[\![\mu\gamma.C]\!]_\eta &= fix(\lambda c \in Cap.\ [\![C]\!]_{\eta[\gamma:=c]}) & [\![\forall\sigma.C]\!]_\eta &= \forall(\lambda v \in Val.\ [\![C]\!]_{\eta[\sigma:=v]})
\end{aligned}
$$

Value types, $[\![\tau]\!]_\eta : 1/2 \cdot W \to_{mon} URel(Val)$

$$
\begin{aligned}
[\![\tau \otimes C]\!]_\eta &= [\![\tau]\!]_\eta \otimes \iota([\![C]\!]_\eta) & [\![0]\!]_\eta &= \lambda w.\emptyset \\
[\![1]\!]_\eta &= \lambda w.\mathbb{N} \times \{\langle\rangle\} & [\![\mathsf{int}]\!]_\eta &= \lambda w.\mathbb{N} \times \{\underline{n} \mid n \in \mathbb{Z}\} \\
[\![[\sigma]]\!]_\eta &= \lambda w.\mathbb{N} \times \{\eta(\sigma)\} & [\![\tau_1 + \tau_2]\!]_\eta &= [\![\tau_1]\!]_\eta + [\![\tau_2]\!]_\eta \\
[\![\tau_1 \times \tau_2]\!]_\eta &= [\![\tau_1]\!]_\eta \times [\![\tau_2]\!]_\eta & [\![\chi_1 \to \chi_2]\!]_\eta &= [\![\chi_1]\!]_\eta \to [\![\chi_2]\!]_\eta \\
[\![\alpha]\!]_\eta &= \eta(\alpha) & [\![\mu\alpha.\tau]\!]_\eta &= fix(\lambda g \in VT.\ [\![\tau]\!]_{\eta[\alpha:=g]}) \\
[\![\forall\sigma.\tau]\!]_\eta &= \forall(\lambda v \in Val.\ [\![\tau]\!]_{\eta[\sigma:=v]})
\end{aligned}
$$

Memory types, $[\![\theta]\!]_\eta : 1/2 \cdot W \to_{mon} URel(Val \times Heap)$

$$
\begin{aligned}
[\![\theta \otimes C]\!]_\eta &= [\![\theta]\!]_\eta \otimes \iota([\![C]\!]_\eta) & [\![\theta_1 + \theta_2]\!]_\eta &= [\![\theta_1]\!]_\eta + [\![\theta_2]\!]_\eta \\
[\![\tau]\!]_\eta &= \lambda w.\{(k,(v,h)) \mid (k,v) \in [\![\tau]\!]_\eta\, w\} & [\![\theta_1 \times \theta_2]\!]_\eta &= [\![\theta_1]\!]_\eta \times [\![\theta_2]\!]_\eta \\
[\![\mathsf{ref}\ \theta]\!]_\eta &= ref\ [\![\theta]\!]_\eta & [\![\theta * C]\!]_\eta &= [\![\theta]\!]_\eta * [\![C]\!]_\eta \\
[\![\beta]\!]_\eta &= \eta(\beta) & [\![\exists\sigma.\theta]\!]_\eta &= \exists(\lambda v \in Val.\ [\![\theta]\!]_{\eta[\sigma:=v]}) \\
[\![\mu\beta.\theta]\!]_\eta &= fix(\lambda g \in MT.\ [\![\theta]\!]_{\eta[\beta:=g]}) & [\![\forall\sigma.\theta]\!]_\eta &= \forall(\lambda v \in Val.\ [\![\theta]\!]_{\eta[\sigma:=v]})
\end{aligned}
$$

Value contexts, $[\![\Delta]\!]_\eta : 1/2 \cdot W \to_{mon} URel(Env)$

$$
\begin{aligned}
[\![\Delta \otimes C]\!]_\eta &= [\![\Delta]\!]_\eta \otimes \iota([\![C]\!]_\eta) \\
[\![\varnothing]\!]_\eta &= \lambda w.\mathbb{N} \times \{[]\} \\
[\![\Delta, x{:}\tau]\!]_\eta &= \lambda w.\{(k, \rho[x \mapsto v]) \mid (k,\rho) \in [\![\Delta]\!]_\eta\, w \ \wedge\ (k,v) \in [\![\tau]\!]_\eta\, w\}
\end{aligned}
$$

Affine contexts, $[\![\Gamma]\!]_\eta : 1/2 \cdot W \to_{mon} URel(Env \times Heap)$

$$
\begin{aligned}
[\![\Gamma \otimes C]\!]_\eta &= [\![\Gamma]\!]_\eta \otimes \iota([\![C]\!]_\eta) \\
[\![\varnothing]\!]_\eta &= \lambda w.\mathbb{N} \times (\{[]\} \times Heap) \\
[\![\Gamma, x{:}\chi]\!]_\eta\, w &= \lambda w.\{(k, (\rho[x \mapsto v], h \cdot h')) \mid \\
&\qquad (k,(\rho,h)) \in [\![\Gamma]\!]_\eta\, w \ \wedge\ (k,(v,h')) \in [\![\chi]\!]_\eta\, w\} \\
[\![\Gamma * C]\!]_\eta &= [\![\Gamma]\!]_\eta * [\![C]\!]_\eta
\end{aligned}
$$

Fig. 11. Interpretation of capabilities and types

Proof. We first prove the claim that for all C, C',

$$\imath\,[\![C \circ C']\!] = \imath\,[\![C]\!] \circ \imath\,[\![C']\!]\,.$$

It suffices to show

$$[\![C \circ C']\!]_\eta\, w = \imath^{-1}(\imath\,[\![C]\!]_\eta \circ \imath\,[\![C']\!]_\eta)(w)$$

for all η and w, and this follows from the defining equation for $\circ$:

$$\begin{aligned}\imath^{-1}(\imath\,[\![C]\!]_\eta \circ \imath\,[\![C']\!]_\eta)(w) &= \imath^{-1}(\imath\,[\![C]\!]_\eta)(\imath\,[\![C']\!]_\eta \circ w) * \imath^{-1}(\imath\,[\![C']\!]_\eta)(w)\\ &= ([\![C]\!]_\eta \otimes \imath\,[\![C']\!]_\eta)(w) * [\![C']\!]_\eta\,(w)\\ &= [\![C \otimes C' * C']\!]_\eta\,(w) = [\![C \circ C']\!]_\eta\,(w).\end{aligned}$$

To prove the lemma, it now suffices to prove

$$\imath\,[\![C_1 \circ (C_2 \circ C_3)]\!] = \imath\,[\![(C_1 \circ C_2) \circ C_3]\!]\,,$$

but by the above claim, this is a consequence of the associativity of $\circ$ on X. $\square$

Most of the remaining equations in Figure 5 (as well as other equivalences that appear in Charguéraud and Pottier (2008) and Pottier (2008)) are easy consequences of the pointwise definition of the operations in Definition 18. We will now consider the distribution axiom for arrow types, which is more involved.

Lemma 21. For all χ_1, χ_2 and C,

$$[\![(\chi_1 \to \chi_2) \otimes C]\!] = [\![(\chi_1 \circ C) \to (\chi_2 \circ C)]\!]\,.$$

Proof. The lemma follows from the following claim:

$$\forall g_1, g_2 \in MT.\ \forall c \in Cap.\ (g_1 \to g_2) \otimes \imath(c) = (g_1 \otimes \imath(c) * c) \to (g_2 \otimes \imath(c) * c)$$

We will just prove the inclusion from left to right. For the proof, we let $x \in X$, $k \in \mathbb{N}$ and assume

$$(k, (\mathsf{fun}\, f(y) = t)) \in ((g_1 \to g_2) \otimes \imath(c))(x) = (g_1 \to g_2)(\imath(c) \circ x).$$

We must show that

$$(k, (\mathsf{fun}\, f(y) = t)) \in (g_1 \otimes \imath(c) * c) \to (g_2 \otimes \imath(c) * c).$$

To this end, let $j < k$, $w \in W$, $r \in URel(Heap)$, and suppose

$$\begin{aligned}(j, (v, h)) &\in (g_1 \otimes \imath(c) * c)(x \circ w) * \imath^{-1}(x \circ w)(emp) * r\\ &= g_1(\imath(c) \circ x \circ w) * c(x \circ w) * \imath^{-1}(x \circ w)(emp) * r\\ &= g_1(\imath(c) \circ x \circ w) * \imath^{-1}(\imath(c) \circ x \circ w)(emp) * r.\end{aligned}$$

Then, by assumption,

$$(j, (t[f{:=}\mathsf{fun}\, f(y) = t, y{:=}v], h)) \in \mathscr{E}(g_2 * r)(\imath(c) \circ x \circ w).$$

By unfolding the definition of $\mathscr{E}$, this can be seen to be equivalent to

$$(j, (t[f{:=}\mathsf{fun}\, f(y) = t, y{:=}v], h)) \in \mathscr{E}(g_2 \otimes \imath(c) * c * r)(x \circ w),$$

so

$$(k, (\mathsf{fun}\, f(y) = t)) \in (g_1 \otimes \iota(c) * c) \to (g_2 \otimes \iota(c) * c).$$

The other inclusion is proved similarly. □

We give the semantics of typing judgements next. The semantics of a typing judgement for values simply establishes truth with respect to all worlds w, environments η and indices $k \in \mathbb{N}$:

$$\models (\Delta \vdash v : \tau) \iff \forall \eta.\ \forall w.\ \forall k.\ \forall \rho.\ (k, \rho) \in [\![\Delta]\!]_\eta\, w \ \Rightarrow\ (k, \rho(v)) \in [\![\tau]\!]_\eta\, w.$$

Here $\rho(v)$ means the application of the substitution ρ to v.

The semantics of the typing judgement for expressions mirrors the interpretation of the arrow case for value types in that there is also a quantification over heap predicates $r \in URel(Heap)$ and an existential quantification over $w' \in W$ through the use of $\mathscr{E}$:

$$\begin{aligned}\models\ (\Gamma \Vdash t : \chi) \iff\ & \forall \eta.\ \forall w \in W.\ \forall k.\ \forall \rho.\ \forall h.\ \forall r \in URel(Heap).\\ & (k, (\rho, h)) \in [\![\Gamma]\!]_\eta\, w * \iota^{-1}(w)(emp) * r\\ & \quad \Rightarrow\ (k, (\rho(t), h)) \in \mathscr{E}([\![\chi]\!]_\eta * r)(w).\end{aligned}$$

The universal quantification over worlds w ensures the soundness of the deep frame rule, and the universal quantification over heap predicates r validates the shallow frame rule. The existential quantifier plays an important part in the verification of the anti-frame rule below.

The rest of this section is devoted to proving the soundness of the calculus of capabilities. Note that, in particular, soundness means that a well-typed closed program is safe to execute (does not go wrong).

Theorem 22 (Soundness).

— If $\Delta \vdash v : \tau$, then $\models (\Delta \vdash v : \tau)$.
— If $\Gamma \Vdash t : \chi$, then $\models (\Gamma \Vdash t : \chi)$.

In particular, if $\varnothing \vdash t : \chi$ is a closed program that does not contain any locations, and if $(t \,|\, h) \longmapsto^* (t' \,|\, h')$ where $(t' \,|\, h')$ is irreducible, then t' is a value.

To prove the theorem, we show that each typing rule preserves the truth of judgements. The proof of the frame rules is straightforward.

Lemma 23 (Soundness of the shallow frame rule). Suppose $\models (\Gamma \Vdash t : \chi)$. Then $\models (\Gamma * C \Vdash t : \chi * C)$.

Proof. We assume $\models (\Gamma \Vdash t : \chi)$ and prove $\models (\Gamma * C \Vdash t : \chi * C)$. Let η be an environment, let $w \in W$, $k \in \mathbb{N}$, $r \in URel(Heap)$ and assume

$$(k, (\rho, h)) \in [\![\Gamma * C]\!]_\eta (w) * \iota^{-1}(w)(emp) * r = [\![\Gamma]\!]_\eta (w) * [\![C]\!]_\eta (w) * \iota^{-1}(w)(emp) * r.$$

We can now instantiate the universally quantified r in the assumption $\models (\Gamma \Vdash t : \chi)$ with $[\![C]\!]_\eta (w) * r$ to obtain

$$(k, (\rho(t), h)) \in \mathscr{E}([\![\chi]\!]_\eta * ([\![C]\!]_\eta (w) * r))(w).$$

Since $[\![C]\!]_\eta \in \mathit{Cap}$, we have

$$[\![C]\!]_\eta (w) \subseteq [\![C]\!]_\eta (w \circ w')$$

for any $w' \in W$, and we then get

$$(k, (\rho(t), h)) \in \mathscr{E}([\![\chi * C]\!]_\eta * r)(w)$$

by unfolding the definition of $\mathscr{E}$. □

Lemma 24 (Soundness of the deep frame rule for expressions). Suppose

$$\models (\Gamma \Vdash t : \chi).$$

Then

$$\models (\Gamma \otimes C * C \Vdash t : \chi \otimes C * C).$$

Proof. Assuming $\models (\Gamma \Vdash t : \chi)$, we need to prove

$$\models (\Gamma \otimes C * C \Vdash t : \chi \otimes C * C).$$

Let η be an environment, let $w \in W$, $k \in \mathbb{N}$, $r \in \mathit{URel}(\mathit{Heap})$ and

$$\begin{aligned}(k, (\rho, h)) \in [\![\Gamma \otimes C * C]\!]_\eta (w) * \iota^{-1}(w)(\mathit{emp}) * r \\ = [\![\Gamma]\!]_\eta (\iota([\![C]\!]_\eta) \circ w) * \iota^{-1}(\iota([\![C]\!]_\eta) \circ w)(\mathit{emp}) * r.\end{aligned}$$

Since $[\![C]\!]_\eta \in \mathit{Cap}$, we can instantiate $\models (\Gamma \Vdash t : \chi)$ with the world

$$w' = \iota([\![C]\!]_\eta) \circ w$$

to obtain

$$(k, (\rho(t), h)) \in \mathscr{E}([\![\chi]\!]_\eta * r)(w'),$$

which is equivalent to

$$(k, (\rho(t), h)) \in \mathscr{E}([\![\chi \otimes C * C]\!]_\eta * r)(w).$$

This completes the proof. □

Next we consider the anti-frame rule. Our soundness proof of the anti-frame rule employs the so-called technique of commutative pairs. This idea was already present in Pottier's syntactic proof sketch in Pottier (2008), and has been worked out in more detail in Schwinghammer *et al.* (2010). The intuitive idea is that the pair denoted w_0' and w_1' below can be used to merge two invariants denoted w_0 and w_1 such that all computations described in the first invariant w_0 preserve the second invariant w_1, and *vice versa*.

Lemma 25 (Existence of commutative pairs). For all worlds $w_0, w_1 \in W$, there exist $w_0', w_1' \in W$ such that

$$\begin{aligned} w_0' &= \iota(\iota^{-1}(w_0) \otimes w_1') \\ w_1' &= \iota(\iota^{-1}(w_1) \otimes w_0') \\ w_0 \circ w_1' &= w_1 \circ w_0'. \end{aligned}$$

Proof. We fix $w_0, w_1 \in W$ and consider the function F on $X \times X$ defined by

$$F(x_0', x_1') \;=\; \big(\iota(\iota^{-1}(w_0) \otimes x_1'),\ \iota(\iota^{-1}(w_1) \otimes x_0')\big).$$

F is contractive, since $\otimes$ is contractive in its second argument. Also, F restricts to a function on the non-empty and closed subset $W \times W$ of $X \times X$. Thus, by Banach's fixed-point theorem, F has a unique fixed point $(w_0', w_1') \in W \times W$. This means that

$$\begin{aligned} w_0' &= \iota(\iota^{-1}(w_0) \otimes w_1') \\ w_1' &= \iota(\iota^{-1}(w_1) \otimes w_0'). \end{aligned} \tag{29}$$

Note that these are the first two equalities claimed by this lemma. The remaining claim is $w_0 \circ w_1' = w_1 \circ w_0'$, which can be proved as follows. Let $w \in X$. Then,

$$\begin{aligned}
\iota^{-1}(w_0 \circ w_1')(w) &= \iota^{-1}(w_0)(w_1' \circ w) * \iota^{-1}(w_1')(w) && \text{(by the definition of } \circ) \\
&= (\iota^{-1}(w_0) \otimes w_1')(w) * \iota^{-1}(w_1')(w) && \text{(by the definition of } \otimes) \\
&= \iota^{-1}(w_0')(w) * (\iota^{-1}(w_1) \otimes w_0')(w) && \text{(by (29))} \\
&= \iota^{-1}(w_0')(w) * \iota^{-1}(w_1)(w_0' \circ w) && \text{(by the definition of } \otimes) \\
&= \iota^{-1}(w_1)(w_0' \circ w) * \iota^{-1}(w_0')(w) && \text{(by the commutativity of } *) \\
&= \iota^{-1}(w_1 \circ w_0')(w). && \text{(by the definition of } \circ)
\end{aligned}$$

Then, since w was chosen arbitrarily, we have $\iota^{-1}(w_0 \circ w_1') = \iota^{-1}(w_1 \circ w_0')$, and the claim follows from the injectivity of ι^{-1}. □

Lemma 26 (Soundness of the anti-frame rule). Suppose

$$\models (\Gamma \otimes C \Vdash t : \chi \otimes C * C).$$

Then

$$\models (\Gamma \Vdash t : \chi).$$

Proof. To prove $\models (\Gamma \Vdash t : \chi)$, we let $w \in W$, η be an environment, $r \in URel(Heap)$ and

$$(k, (\rho, h)) \in [\![\Gamma]\!]_\eta\,(w) * \iota^{-1}(w)(emp) * r.$$

We must prove $(k, (\rho(t), h)) \in \mathscr{E}([\![\chi]\!]_\eta * r)(w)$. By Lemma 25,

$$\begin{aligned} w_1 &= \iota(\iota^{-1}(w) \otimes w_2) \\ w_2 &= \iota([\![C]\!]_\eta \otimes w_1) \\ \iota([\![C]\!]_\eta) \circ w_1 &= w \circ w_2 \end{aligned} \tag{30}$$

holds for some worlds w_1, w_2 in W.

First, we find a superset of the precondition $[\![\Gamma]\!]_\eta\,(w) * \iota^{-1}(w)(\mathit{emp}) * r$ in the assumption above by replacing the first two $*$-conjuncts as follows:

$$\begin{aligned}
[\![\Gamma]\!]_\eta\,(w) &\subseteq [\![\Gamma]\!]_\eta\,(w \circ w_2) && \text{(by the monotonicity of } [\![\Gamma]\!]_\eta \text{ and } w_2 \in W)\\
&= [\![\Gamma]\!]_\eta\,(\iota([\![C]\!]_\eta) \circ w_1) && \text{(since } \iota([\![C]\!]_\eta) \circ w_1 = w \circ w_2)\\
&= [\![\Gamma \otimes C]\!]_\eta\,(w_1). && \text{(by the definition of } \otimes)
\end{aligned}$$

$$\begin{aligned}
\iota^{-1}(w)(\mathit{emp}) &\subseteq \iota^{-1}(w)(\mathit{emp} \circ w_2) && \text{(by the monotonicity of } \iota^{-1}(w) \text{ and } w_2 \in W)\\
&= \iota^{-1}(w)(w_2 \circ \mathit{emp}) && \text{(since } \mathit{emp} \text{ is the unit)}\\
&= (\iota^{-1}(w) \otimes w_2)(\mathit{emp}) && \text{(by the definition of } \otimes)\\
&= \iota^{-1}(w_1)(\mathit{emp}). && \text{(since } w_1 = \iota(\iota^{-1}(w) \otimes w_2))
\end{aligned}$$

Thus, by the monotonicity of the separating conjunction, we have

$$(k, (\rho, h)) \in [\![\Gamma]\!]_\eta\,(w) * \iota^{-1}(w)(\mathit{emp}) * r \;\subseteq\; [\![\Gamma \otimes C]\!]_\eta\,(w_1) * \iota^{-1}(w_1)(\mathit{emp}) * r. \tag{31}$$

By the assumed validity of the judgement $\Gamma \otimes C \Vdash t : \chi \otimes C * C$, this entails

$$(k, (\rho(t), h)) \in \mathscr{E}([\![\chi \otimes C * C]\!]_\eta * r)(w_1). \tag{32}$$

We need to show that

$$(k, (\rho(t), h)) \in \mathscr{E}([\![\chi]\!]_\eta * r)(w),$$

so we assume $(\rho(t) \,|\, h) \longmapsto^j (t' \,|\, h')$ for some $j \leqslant k$ such that $(t' \,|\, h')$ is irreducible. From (32) we then obtain

$$(k{-}j, (t', h')) \in \textstyle\bigcup_{w'} [\![\chi \otimes C * C]\!]_\eta\,(w_1 \circ w') * \iota^{-1}(w_1 \circ w')(\mathit{emp}) * r. \tag{33}$$

Now observe that, for any w', we have

$$\begin{aligned}
&[\![\chi \otimes C * C]\!]_\eta\,(w_1 \circ w') * \iota^{-1}(w_1 \circ w')(\mathit{emp})\\
&\qquad = [\![\chi]\!]_\eta\,(\iota([\![C]\!]_\eta) \circ w_1 \circ w') * [\![C]\!]_\eta\,(w_1 \circ w') * \iota^{-1}(w_1 \circ w')(\mathit{emp})\\
&\qquad = [\![\chi]\!]_\eta\,(\iota([\![C]\!]_\eta) \circ w_1 \circ w') * \iota^{-1}(\iota([\![C]\!]_\eta) \circ w_1 \circ w')(\mathit{emp})\\
&\qquad = [\![\chi]\!]_\eta\,(w \circ w_2 \circ w') * \iota^{-1}(w \circ w_2 \circ w')(\mathit{emp})
\end{aligned}$$

since $\iota([\![C]\!]_\eta) \circ w_1 = w \circ w_2$. Setting $w'' \stackrel{\mathit{def}}{=} w_2 \circ w'$, we get that the last line equals

$$[\![\chi]\!]_\eta\,(w \circ w'') * \iota^{-1}(w \circ w'')(\mathit{emp}).$$

Thus, (33) means that $(k{-}j, (t', h'))$ is in

$$\bigcup_{w''} [\![\chi]\!]_\eta\,(w \circ w'') * \iota^{-1}(w \circ w'')(\mathit{emp}) * r,$$

and we are done. □

Remark 27 (Monotonicity). Note that it is in the above proof for the anti-frame rule that we exploited the monotonicity condition of the recursive worlds to establish (31). The monotonicity of $[\![C]\!]$ is also used to prove the shallow frame rule in Lemma 23 (and the first-order frame axiom in Proposition 28 below). However, this is only necessary because of the existential quantifier that is implicitly used in the postcondition through the definition of $\mathscr{E}(\cdot)$. In a system without the anti-frame rule, the quantifier can be dropped from the definition of $\mathscr{E}(\cdot)$ and no monotonicity condition of $[\![C]\!]$ would be needed (Birkedal *et al.* 2011; Schwinghammer *et al.* 2009).

We will omit the proofs for the remaining typing rules. Using the model, we can also show that subtyping is sound. Recall that $\leqslant$ is an inductively defined relation on syntactic type expressions, which is defined by axioms (as shown in Figure 10) and rules that propagate those axioms through type constructors (which are omitted for brevity). We can now show that syntactic subtyping is sound.

Proposition 28 (Soundness of subtyping). The three kinds of subtyping relations are sound. More precisely, for all η and w:

(1) $C \leqslant C'$ implies $[\![C]\!]_\eta\, w \subseteq [\![C']\!]_\eta\, w$.
(2) $\tau \leqslant \tau'$ implies $[\![\tau]\!]_\eta\, w \subseteq [\![\tau']\!]_\eta\, w$.
(3) $\theta \leqslant \theta'$ implies $[\![\theta]\!]_\eta\, w \subseteq [\![\theta']\!]_\eta\, w$.

Proof. The three statements are proved simultaneously by induction on the derivation of the subtyping judgement in question. We must show that the axioms in Figure 10 hold with respect to the interpretation given in Figure 11, and that all of the inference rules that define the subtyping judgements preserve these inclusions. We will just show three sample cases:

— **Axiom** (25) **is sound:**

We have to show for all η and $w \in W$ that

$$[\![\chi_1 \to \chi_2]\!]_\eta\, w \subseteq [\![(\chi_1 * C) \to (\chi_2 * C)]\!]_\eta\, w.$$

We assume $(k, \mathsf{fun}\, f(x) = t) \in [\![\chi_1 \to \chi_2]\!]_\eta\, w$. To see that $(k, \mathsf{fun}\, f(x) = t)$ is also in the set $[\![(\chi_1 * C) \to (\chi_2 * C)]\!]_\eta\, w$, suppose that $j < k$, $w_0 \in W$ and $r \in URel(Heap)$, and let

$$\begin{aligned}(j,(v,h)) \in [\![\chi_1 * C]\!]_\eta\,(w \circ w_0) * \iota^{-1}(w \circ w_0)(emp) * r \\ = [\![\chi_1]\!]_\eta\,(w \circ w_0) * \iota^{-1}(w \circ w_0)(emp) * (r * [\![C]\!]_\eta\,(w \circ w_0)).\end{aligned}$$

We must show that

$$(j,(t[f{:=}\mathsf{fun}\, f(x) = t, x{:=}v], h)) \in \mathscr{E}([\![\chi_2 * C]\!]_\eta * r)(w \circ w_0).$$

So we assume that

$$(t[f{:=}\mathsf{fun}\, f(x) = t, x{:=}v] \mid h) \longmapsto^i (t' \mid h')$$

for some $i \leqslant j$ and some irreducible configuration $(t' \mid h')$. By unfolding the definition of $[\![\chi_1 \to \chi_2]\!]_\eta\, w$, we obtain

$$(j,(t[f{:=}\mathsf{fun}\, f(x) = t, x{:=}v], h)) \in \mathscr{E}([\![\chi_2]\!]_\eta * (r * [\![C]\!]_\eta\,(w \circ w_0)))(w \circ w_0),$$

and hence that there exists $w_1 \in W$ such that

$$(j - i,(t', h')) \in [\![\chi_2]\!]_\eta\,(w \circ w_0 \circ w_1) * \iota^{-1}(w \circ w_0 \circ w_1)(emp) * r * [\![C]\!]_\eta\,(w \circ w_0).$$

Since

$$w \circ w_0 \sqsubseteq w \circ w_0 \circ w_1$$

entails

$$[\![C]\!]_\eta\,(w \circ w_0) \subseteq [\![C]\!]_\eta\,(w \circ w_0 \circ w_1),$$

by the monotonicity of $[\![C]\!]_\eta$, we get

$$(j-i,(t',h')) \in [\![\chi_2 * C]\!]_\eta\,(w \circ w_0 \circ w_1) * \iota^{-1}(w \circ w_0 \circ w_1)(\mathit{emp}) * r,$$

which yields

$$(j,(t[f{:=}\mathsf{fun}\,f(x)=t, x{:=}v],h)) \in \mathscr{E}([\![\chi_2 * C]\!]_\eta * r)(w \circ w_0).$$

— **Axiom 26 is sound:**

We have to prove for all η and $w \in W$, $[\![C]\!]_\eta\, w \subseteq [\![\emptyset]\!]_\eta\, w$, but this follows simply from the definition $[\![\emptyset]\!]_\eta\, w = \mathbb{N} \times \mathit{Heap}$.

— **The rule for covariant subtyping of $\otimes$, concluding $\tau \otimes C \leqslant \tau' \otimes C$ from $\tau \leqslant \tau'$, is sound:**

We assume that

$$[\![\tau]\!]_\eta\, w \subseteq [\![\tau']\!]_\eta\, w$$

holds for all η and $w \in W$. Then we have to show that

$$[\![\tau \otimes C]\!]_\eta\, w \subseteq [\![\tau' \otimes C]\!]_\eta\, w$$

for all η and $w \in W$.

By definition,

$$[\![\tau \otimes C]\!]_\eta\, w = [\![\tau]\!]_\eta\,(\iota\,[\![C]\!]_\eta \circ w)$$
$$[\![\tau' \otimes C]\!]_\eta\, w = [\![\tau']\!]_\eta\,(\iota\,[\![C]\!]_\eta \circ w).$$

Thus, the statement follows by instantiating the universally quantified world in the assumption by $\iota\,[\![C]\!]_\eta \circ w$. □

The soundness of the subsumption rules in Figures 8 and 9 is an immediate consequence of Proposition 28.

8. Generalised frame and anti-frame rules

The frame and anti-frame rules allow the hiding of *invariants*. However, to hide uses of local state, say for a function, it is, in general, not enough just to allow the hiding of global invariants that are preserved across arbitrary sequences of calls and returns. For instance, consider the function f with local reference cell r:

$$\mathsf{let}\ r = \mathsf{ref}\ 0\ \mathsf{in}\ \mathsf{fun}\,f(g) = (\mathit{inc}(r); g\langle\rangle; \mathit{dec}(r)). \qquad (34)$$

If we write $\mathsf{int}\ n$ for the singleton integer type containing n, we may wish to hide the capability $I = \{\sigma : \mathsf{ref}\,(\mathsf{int}\ 0)\}$ to capture the intuition that the cell $r : [\sigma]$ stores 0 on

$$\frac{\Gamma \Vdash t : \chi}{\Gamma \otimes I * I\, i \Vdash t : \exists j \geqslant i. (\chi \otimes I) * I\, j}\ \textsc{Generalised frame} \qquad \frac{\Gamma \otimes I \Vdash t : \exists i. (\chi \otimes I) * I\, i}{\Gamma \Vdash t : \chi}\ \textsc{Generalised anti-frame}$$

Fig. 12. Generalised frame and anti-frame rules

termination. However, there could well be re-entrant calls to f such that $\{\sigma : \mathsf{ref}\,(\mathsf{int}\ 0)\}$ is not an invariant for those calls.

Thus Pottier (2009a) proposed two extensions to the anti-frame rule that allow for hiding of families of invariants. The first idea is that each invariant in the family is a *local* invariant that holds for one level of the recursive call of a function. This extension allows us to hide 'well-bracketed' (Dreyer *et al.* 2010) uses of local state. For instance, the $\mathbb{N}$-indexed family of invariants $I\, n = \{\sigma : \mathsf{ref}\,(\mathsf{int}\ n)\}$ can be used for (34) – see the examples in Pottier (2009a). The second idea is to allow each local invariant to *evolve* in some monotonic fashion, which allows us to hide even more uses of local state. For instance, for f defined by

$$\mathsf{let}\ r = \mathsf{ref}\ 1\ \mathsf{in}\ \mathsf{fun}\, f(g) = (\mathsf{set}\,\langle r, 0\rangle\, ; g\langle\rangle ; \mathsf{set}\,\langle r, 1\rangle\, ; g\langle\rangle),$$

we may wish to capture the fact that the cell $r : [\sigma]$ stores 1 after $f(g)$ returns. Intuitively, this holds since the calls to g may at most bump up the value of r from 0 to 1 (through recursive calls to f), and this fact can be captured in the type system by considering the $\{0, 1\}$-indexed family of invariants $I\, n = \{\sigma : \mathsf{ref}\,(\mathsf{int}\ n)\}$ once we allow for the fact that calls with $I\, i$ may return with $I\, j$ for $j \geqslant i$. The idea is related to the notion of evolving invariants for local state in recent work on reasoning about contextual equivalence (Ahmed *et al.* 2009; Dreyer *et al.* 2010).

In summary, we want to allow the hiding of a family of capabilities $(I\, i)_{i \in \kappa}$ indexed over a preordered set $(\kappa, \leqslant)$. The preorder is used to capture the fact that the local invariants can evolve in a monotonic fashion, as expressed in the new definition of the action of $\otimes$ on function types (note that I on the right-hand side of $\otimes$ now has kind $\kappa \to \textsc{cap}$):

$$(\chi_1 \to \chi_2) \otimes I \;\; = \;\; \forall i.\ \big((\chi_1 \otimes I) * I\, i \to \exists j \geqslant i.\ ((\chi_2 \otimes I) * I\, j)\big). \tag{35}$$

Observe how this definition captures the intuitive idea: if the invariant $I\, i$ holds when the function is called, then, upon return, we know that an invariant $I\, j$ (for $j \in \kappa$, $j \geqslant i$) holds. Different recursive calls may use different local invariants due to the quantification over i. The generalised frame and anti-frame rules are given in Figure 12.

We now show how to extend our model of the type and capability calculus to accommodate hiding of more expressive families of invariants like these. Naturally, the first step is to refine our notion of world, since the worlds are used to describe hidden invariants.

8.1. *Generalised recursive worlds and generalised world extension*

Suppose $\mathscr{K}$ is a (small) collection of preordered sets. We write $\mathscr{K}^*$ for the finite sequences over $\mathscr{K}$ and ε for the empty sequence, and use juxtaposition to denote concatenation.

For convenience, we will sometimes identify a sequence $\alpha = \kappa_1, \ldots, \kappa_n$ over $\mathscr{K}$ with the preorder $\kappa_1 \times \cdots \times \kappa_n$. As in Section 6, we define the worlds for the Kripke model in two steps, starting from an equation without any monotonicity requirements[†]: **CBUlt** has all non-empty coproducts; and there is a unique solution to the two equations

$$\begin{aligned} X &\cong \sum_{\alpha \in \mathscr{K}^*} X_\alpha \\ X_{\kappa_1,\ldots,\kappa_n} &= (\kappa_1 \times \cdots \times \kappa_n) \to \left(\tfrac{1}{2} \cdot X \to URel(Heap)\right), \end{aligned} \tag{36}$$

with isomorphism $\iota : \sum_{\alpha \in \mathscr{K}^*} X_\alpha \to X$ in **CBUlt**, where each $\kappa \in \mathscr{K}$ is equipped with the discrete metric. Each X_α consists of the α-indexed families of (world-dependent) predicates so that, in comparison to Section 6, X consists of all these families rather than individual predicates.

Note that, by the definition of the metric on X, if $x \overset{n}{=} x'$ holds for $n > 0$ and $x = \iota\langle \alpha, g\rangle$ and $x' = \iota\langle \alpha', g'\rangle$, then $\alpha = \alpha'$ and $g\, i \overset{n}{=} g'\, i$ for all $i \in \alpha$.

The composition operation $\circ : X \times X \to X$ is now given by $x_1 \circ x_2 = \iota(\langle \alpha_1\alpha_2, g\rangle)$, where $\langle \alpha_i, g_i\rangle = \iota^{-1}(x_i)$, and where $g \in X_{\alpha_1\alpha_2}$ is defined by

$$g(i_1 i_2)(x) \;=\; g_1(i_1)(x_2 \circ x) * g_2(i_2)(x)$$

for $i_1 \in \alpha_1$, $i_2 \in \alpha_2$. That is, the combination of an α_1-indexed family g_1 and an α_2-indexed family g_2 is a family g over $\alpha_1\alpha_2$, but there is no interaction between the index components i_1 and i_2: they are concerned with disjoint regions of the heap. The composition operation is defined as the fixed point of a contractive function as in Lemma 10, and it can be shown to be associative and has a left and right unit given by $emp = \iota(\langle \varepsilon, I\rangle)$. For $g : \frac{1}{2} \cdot X \to URel(A)$, we define the extension operation $(g \otimes x)(x') = g(x \circ x')$

8.2. *Generalised hereditarily monotone recursive worlds*

We will proceed as in Section 6 and carve out a subset of recursive worlds that satisfy a monotonicity condition.

To prove the soundness of the anti-frame rule, and, more specifically, to establish the existence of commutative pairs, we need to know that the order in which the invariant families appear is irrelevant for the semantics of types and capabilities. The requirement is made precise by considering a partial equivalence relation $\sim$ on X, where

$$\iota(\langle \alpha_1\alpha_2, g\rangle) \sim \iota(\langle \alpha_2\alpha_1, h\rangle)$$

holds if

$$g(i_1 i_2)(x_1) = h(i_2 i_1)(x_2)$$

for all $i_1 \in \alpha_1$, $i_2 \in \alpha_2$ and $x_1 \sim x_2$, and insisting that semantic operations respect this relation. Note that the relation $\sim$ is recursive; we define it as the fixed point of a function Ψ on the non-empty and closed subsets of $X \times X$.

[†] We believe that a variant of the inverse-limit construction in Section 5 could also be used to construct the worlds, but we have not checked all the details.

Definition 29. Let $\Psi : \mathscr{R}(X \times X) \to \mathscr{R}(X \times X)$ be defined as follows. For all $x, y \in X$ where $x = \iota\langle \alpha, g \rangle$ and $y = \iota\langle \beta, h \rangle$, we have $(x, y) \in \Psi(R)$ if and only if:

— there exists $n \in \mathbb{N}$ and a permutation π of $1, \dots, n$ such that $\alpha = \alpha_1 \dots \alpha_n$ and $\beta = \alpha_{\pi(1)} \dots \alpha_{\pi(n)}$; and

— for all $i_1 \in \alpha_1, \dots, i_n \in \alpha_n$ and all $z, z' \in X$, if $(z, z') \in R$, then $g(i_1 \dots i_n)(z) = h(i_{\pi(1)} \dots i_{\pi(n)})(z')$.

The function Ψ is contractive, and we define $\sim\, \subseteq X \times X$ as its unique fixed point in $URel(X \times X)$ by the Banach fixed-point theorem.

Lemma 30. $\sim$ is a partial equivalence relation on X:

(1) $x \sim y$ implies $y \sim x$.
(2) $x \sim y$ and $y \sim z$ implies $x \sim z$.

Proof.

(1) Since $(\sim_{[n]})_n$ is a Cauchy chain in $\mathscr{R}(X \times X)$ with limit $\sim$ given as the intersection of the $\sim_{[n]}$, part (1) of the lemma follows from the claim

$$\forall n \in \mathbb{N}.\ \forall xy \in X.\ x \sim y \ \Rightarrow\ (y, x) \in \sim_{[n]},$$

which is proved by induction on n:

— For $n = 0$, the result is immediate since $\sim_{[0]} = X \times X$.

— For $n > 0$, we let $x \sim y$. For simplicity, we assume

$$x = \iota\langle \alpha_1\alpha_2, p \rangle$$
$$y = \iota\langle \alpha_2\alpha_1, q \rangle.$$

To prove $(y, x) \in \sim_{[n]}$, it suffices to show that $y' \sim x'$ holds for

$$y' = \iota\langle \alpha_2\alpha_1, q' \rangle$$
$$x' = \iota\langle \alpha_1\alpha_2, p' \rangle$$

with

$$q'(i_2i_1)(z) = q(i_2i_1)(z)_{[n]}$$
$$p'(i_1i_2)(z) = p(i_1i_2)(z)_{[n]}$$

since $(y, x) \overset{n}{=} (y', x')$. To this end, we let $i_2 \in \alpha_2$, $i_1 \in \alpha_1$, and suppose that $z \sim z'$. So we must prove $q'(i_2i_1)(z) = p'(i_1i_2)(z')$. By the induction hypothesis, $(z', z) \in \sim_{[n-1]}$, that is, there exists $u' \sim u$ with

$$u' \overset{n-1}{=} z'$$
$$u \overset{n-1}{=} z$$

in X. Note that this means $u' \overset{n}{=} z'$ and $u \overset{n}{=} z$ holds in $\frac{1}{2} \cdot X$. Thus,

$$q(i_2i_1)(z) \overset{n}{=} q(i_2i_1)(u) = p(i_1i_2)(u') \overset{n}{=} p(i_1i_2)(z')$$

by the non-expansiveness of p and q, and by the assumption $x \sim y$. It then follows that

$$q'(i_2i_1)(z) = q(i_2i_1)(z)_{[n]} = p(i_1i_2)(u')_{[n]} = p'(i_1i_2)(z'),$$

that is, we have shown $y' \sim x'$.

(2) We can use a similar argument to prove that for all n, we have $x \sim y$ and $y \sim z$ implies $(x, z) \in \sim_{[n]}$.

The composition operation respects this partial equivalence relation.

Lemma 31. If $x \sim x'$ and $y \sim y'$, then $x \circ y \sim x' \circ y'$.

Proof (sketch). The prof is similar to the proof of Lemma 14. We prove by induction that for all $n \in \mathbb{N}$, if $x \sim x'$ and $y \sim y'$, then $(x \circ y, x' \circ y') \in \sim_{[n]}$. We then use the fact that $\sim$ is the intersection of all the $\sim_{[n]}$. □

We will now define the hereditarily monotone worlds, and ensure that these worlds w respect $\sim$ by requiring that they be self-related. The set $W \subseteq X$ of these worlds is again defined as a fixed point of a contractive function on the closed and non-empty subsets of X.

Definition 32 (Generalised hereditarily monotone worlds). Let $\Phi : \mathscr{R}(X) \to \mathscr{R}(X)$ be defined as follows. For all $w \in X$ where $w = \imath\langle\alpha, g\rangle$, $w \in \Phi(R)$ if and only if:

— $w \sim w$; and
— for all $i \in \alpha$ and all $w_1, w_2 \in R$, we have $g(i)(w_1) \subseteq g(i)(w_1 \circ w_2)$.

The function Φ is contractive, and we define the hereditarily monotone functions $W = \mathit{fix}(\Phi) = \Phi(W)$ by the Banach fixed-point theorem.

Using Lemmas 30 and 31, it is not difficult to see that W is closed under the relation $\sim$. Moreover, as in Section 6, the composition operation restricts to the subset of hereditary monotone worlds.

Lemma 33. If $w_1, w_2 \in W$, then $w_1 \circ w_2 \in W$.

Proof (sketch). As in the proof of Lemma 14, we show that $x, y \in W$ implies $x \circ y \in W_{[n]}$ for all $n \in \mathbb{N}$ by induction on n. Lemma 31 is then used to show the additional requirement that the composition of $x, y \in W$ is self-related, so $x \circ y \sim x \circ y$. □

8.3. *Semantics of capabilities and types.*

The semantic domains for the interpretation of capabilities and types with respect to the generalised worlds now consist of the world-dependent functions that are both monotonic (with respect to the generalised hereditarily monotone worlds) and respect the relation $\sim$. More precisely, for a preordered set A, we define $\frac{1}{2} \cdot W \to_{mon} \mathit{URel}(A)$ to consist of all those $g : \frac{1}{2} \cdot X \to \mathit{URel}(A)$ where:

— $\forall x, x' \in X.\ x \sim x' \ \Rightarrow\ g(x) = g(x')$.
— $\forall w_1, w_2 \in W.\ g(w_1) \subseteq g(w_1 \circ w_2)$.

Then we write

$$\begin{aligned} Cap &= \tfrac{1}{2}\cdot W \to_{mon} URel(Heap) \\ VT &= \tfrac{1}{2}\cdot W \to_{mon} URel(Val) \\ MT &= \tfrac{1}{2}\cdot W \to_{mon} URel(Val \times Heap). \end{aligned}$$

Note that with this definition, $g \in \kappa \to Cap$ if and only if $\iota(\langle \kappa, g\rangle) \in W$.

To define the interpretation of types, we first consider the following extension of memory types from values to expressions. Compared with the corresponding Definition 16 in Section 7, the extension now depends on the parameter $i \in \alpha$.

Definition 34 (Expression typing). Let f be in $\frac{1}{2}\cdot W \to_{mon} URel(Val \times Heap)$. Let $x \in X$ and $\langle \alpha, p\rangle = \iota^{-1}(x)$. Let $i \in \alpha$. Then $\mathscr{E}(f,x,i) \subseteq Exp \times Heap$ is defined by $(k,(t,h)) \in \mathscr{E}(f,x,i)$ if and only if

$$\begin{aligned} &\forall j \leqslant k, t', h'.\ (t \mid h) \longmapsto^{j} (t' \mid h') \ \wedge\ (t' \mid h') \text{ irreducible} \\ &\qquad\qquad \Rightarrow\ (k-j,(t',h')) \in \bigcup_{w\in W,\, \langle \alpha\beta, q\rangle = \iota^{-1}(x\circ w),\, i_1 \geqslant i,\, i_2\in\beta} f(x\circ w) * q(i_1 i_2)(emp). \end{aligned}$$

This definition is well behaved in the sense that: $\mathscr{E}(f,x,i) \subseteq Exp \times Heap$ is a uniform subset (with respect to the discrete order on $Exp \times Heap$); it is non-expansive as a function in x; and $x \sim x'$ implies $\mathscr{E}(f,x,i) = \mathscr{E}(f,x',i')$ for a suitable reordering i' of the parameters i.

Corresponding to the distribution axiom (35), the interpretation of arrow types bakes in the property that state changes on local state are captured by the local invariants: given $x \in X$, $(k, \mathsf{fun}\, f(y) = t) \in (f_1 \to f_2)(x)$ if and only if

$$\begin{aligned} &\forall j < k.\ \forall w \in W \text{ where } \iota^{-1}(x\circ w) = \langle \alpha, p\rangle.\ \forall r \in URel(Heap).\ \forall i \in \alpha.\ \forall v, h. \\ &\qquad (j,(v,h)) \in f_1(x\circ w) * p(i)(emp) * r \ \Rightarrow \\ &\qquad\qquad (j, t[f{:=}\mathsf{fun}\, f(y) = t, y{:=}v], h)) \in \mathscr{E}(f_2 * r, x\circ w, i). \end{aligned}$$

Semantic operations corresponding to the other capability and type constructors can be defined analogously to Definition 18. It is easy to see that these operations respect the relation $\sim$. In fact, the only case that makes direct use of the parameter $x \in W$ is the case of arrow types, where one quantifies universally over the elements of all instances of its precondition p and (through $\mathscr{E}$) existentially over the elements of instances of its postcondition q; by the definition of $\sim$, none of these instances depends on a reordering of the parameter.

As in Section 7 (semantic variants of) the distribution axioms for generalised invariants can be justified with respect to these operations. In particular, the axiom (35) holds since, given $c \in \kappa \to Cap$ and setting $w \overset{def}{=} \iota(\langle \kappa, c\rangle)$,

$$(f_1 \to f_2) \otimes w = \forall_{i\in\kappa}\big(((f_1 \otimes w) * c\, i) \to \exists_{j\geqslant i}((f_2 \otimes w) * c\, j)\big),$$

where $\forall$ and $\exists$ denote the pointwise intersection and union of world-indexed uniform predicates.

The semantics of value judgements $\Delta \vdash v : \tau$ looks like before. The semantics of the expression typing judgement mirrors the new interpretation of arrow types in the sense that there is now also a universal quantification over all possible instances i of the invariant family p represented by a world $w \in W$:

$$\begin{aligned} \models (\Gamma \Vdash t : \chi) \iff & \forall \eta.\, \forall w \in W \text{ where } w = \langle \alpha, p \rangle.\, \forall k \in \mathbb{N}. \\ & \forall i \in \alpha.\, \forall r \in URel(Heap). \forall (k, (\rho, h)) \in [\![\Gamma]\!]_\eta\, w * p(i)(emp) * r. \\ & \qquad (k, (\rho(t), h)) \in \mathscr{E}([\![\chi]\!]_\eta * r, w, i). \end{aligned}$$

We can now prove the soundness of the generalised rules.

Theorem 35 (Soundness). The generalised frame and anti-frame rules are sound.

In particular, this theorem shows that all the reasoning about the use of local state in the (non-trivial) examples considered in Pottier (2009a) is sound.

Proof (sketch). The proof for the generalised frame rule is similar to the proof of Lemma 24.

The soundness proof for the generalised anti-frame rule rests again on the existence of commutative pairs. Compared with the earlier Lemma 25, however, we can only prove a variant that states that commutativity holds up to the relation $\sim$. Let $w_0, w_1 \in W$ be families indexed over α_0 and α_1, that is, $\iota^{-1}(w_0) = \langle \alpha_0, p_0 \rangle$ and $\iota^{-1}(w_1) = \langle \alpha_1, p_1 \rangle$ for some p_0 and p_1. Then there exist $w_0', w_1' \in W$ such that

$$\begin{aligned} w_0' &= \iota \langle \alpha_0, \lambda i.(p_0\, i) \otimes w_1' \rangle \\ w_1' &= \iota \langle \alpha_1, \lambda i.(p_1\, i) \otimes w_0' \rangle \end{aligned}$$

and

$$w_0 \circ w_1' \sim w_1 \circ w_0.$$

Since we insisted that the interpretations of types and capabilities respect $\sim$, this variant is sufficient to prove the soundness of the generalised anti-frame rule analogously to the proof of Lemma 26. □

9. Conclusion and future work

We have developed a soundness proof of the frame and anti-frame rules in the expressive type and capability system of Charguéraud and Pottier by constructing a Kripke model of the system. For our model, we have presented two novel approaches to construct the recursively defined set of worlds[†]. The first approach is a (tedious) construction of an inverse limit in the category of complete, 1-bounded ultrametric spaces. In the second approach, we define the worlds as a recursive subset of a recursively defined metric space. This construction is simpler than the inverse limit construction, but requires an

[†] An interesting challenge would be to find a general existence theorem for solutions of recursive domain equations that can deal with the recursive monotonic worlds.

additional argument to show that the semantic operations restrict to this subset. We have demonstrated that this approach generalises, by extending the model to show the soundness of Pottier's generalised frame and anti-frame rules. More generally, we believe that the recursive worlds constructed in Sections 5 and 6 can be used, possibly with variations, to model various type system and program logics with hidden (higher-order) state.

Future work includes exploring some of the orthogonal extensions of the basic type and capability system that have been proposed in the literature such as group regions (Charguéraud and Pottier 2008) and fates and predictions (Pilkiewicz and Pottier 2011). The model we have presented suggests we include separation logic assertions in the syntax of capabilities, and it would be interesting to work out such a program logic in detail.

Recently, Pottier has given an alternative soundness proof for a slightly different language, which includes group regions as well as the anti-frame rule, but does not include the generalised frame and anti-frame rules. This proof is based on progress and preservation properties, and has been formalised in the Coq proof assistant (Pottier 2011). While we have not attempted a formalisation of our model, we believe that this is possible based on the results in Benton *et al.* (2010).

References

Ahmed, A., Dreyer, D. and Rossberg, A. (2009) State-dependent representation independence. In: *Proceedings of POPL* 340–353.

America, P. and Rutten, J. J. M. M. (1989) Solving reflexive domain equations in a category of complete metric spaces. *Journal of Computer and System Sciences* **39** (3) 343–375.

Appel, A. W. and McAllester, D. A. (2001) An indexed model of recursive types for foundational proof-carrying code. *ACM Transactions on Programming Languages and Systems* **23** (5) 657–683.

Benton, N., Birkedal, L., Kennedy, A. and Varming, C. (2010) Formalizing domains, ultrametric spaces and semantics of programming languages (draft).

Biering, B., Birkedal, L. and Torp-Smith, N. (2007) BI-hyperdoctrines, higher-order separation logic, and abstraction. *ACM Transactions on Programming Languages and Systems* **29** (5).

Birkedal, L., Reus, B., Schwinghammer, J. and Yang, H. (2008) A simple model of separation logic for higher-order store. In: *Proceedings of ICALP* 348–360.

Birkedal, L., Reus, B., Schwinghammer, J., Støvring, K., Thamsborg, J. and Yang, H. (2011) Step-indexed Kripke models over recursive worlds. In: *Proceedings of POPL* 119–132.

Birkedal, L., Støvring, K. and Thamsborg, J. (2009) Realizability semantics of parametric polymorphism, general references, and recursive types. In: *Proceedings of FOSSACS* 456–470.

Birkedal, L., Støvring, K. and Thamsborg, J. (2010) The category-theoretic solution of recursive metric-space equations. *Theoretical Computer Science* **411** (47) 4102–4122.

Birkedal, L., Torp-Smith, N. and Yang, H. (2006) Semantics of separation-logic typing and higher-order frame rules for Algol-like languages. *Logical Methods in Computer Science* **2** (5:1).

Charguéraud, A. and Pottier, F. (2008) Functional translation of a calculus of capabilities. In: *Proceedings of ICFP* 213–224.

Dreyer, D., Neis, G. and Birkedal, L. (2010) The impact of higher-order state and control effects on local relational reasoning. In: *Proceedings of ICFP*.

Gotsman, A., Berdine, J., Cook, B., Rinetzky, N. and Sagiv, M. (2007) Local reasoning for storable locks and threads. Technical Report MSR-TR-2007-39, Microsoft Research.

Hobor, A., Appel, A. W. and Zappa Nardelli, F. (2008) Oracle semantics for concurrent separation logic. In: Proceedings of ESOP. *Springer-Verlag Lecture Notes in Computer Science* **4960** 353–367.

Levy, P. B. (2002) Possible world semantics for general storage in call-by-value. In: *Proceedings of CSL* 232–246.

Nanevski, A., Ahmed, A., Morrisett, G. and Birkedal, L. (2007) Abstract predicates and mutable ADTs in Hoare type theory. In: *Proceedings of ESOP* 189–204.

O'Hearn, P. W. (2007) Resources, concurrency and local reasoning. *Theoretical Computer Science* **375** (1-3) 271–307.

O'Hearn, P. W., Yang, H. and Reynolds, J. C. (2004) Separation and information hiding. In: *Proceedings of POPL* 268–280.

Parkinson, M. and Bierman, G. (2005) Separation logic and abstraction. In: *Proceedings of POPL* 247–258.

Parkinson, M. and Bierman, G. (2008) Separation logic, abstraction and inheritance. In: *Proceedings of POPL* 75–86.

Pierce, B. C. (2002) *Types and Programming Languages*, MIT Press.

Pilkiewicz, A. and Pottier, F. (2011) The essence of monotonic state. In: *Proceedings of TLDI* 73–86.

Pitts, A. M. (1996) Relational properties of domains. *Information and Computation* **127** (2) 66–90.

Pottier, F. (2008) Hiding local state in direct style: a higher-order anti-frame rule. In: *Proceedings of LICS* 331–340.

Pottier, F. (2009a) Generalizing the higher-order frame and anti-frame rules. Unpublished note, available at `http://gallium.inria.fr/~fpottier`.

Pottier, F. (2009b) Three comments on the anti-frame rule. Unpublished note, available at `http://gallium.inria.fr/~fpottier`.

Pottier, F. (2011) Syntactic soundness proof of a type-and-capability system with hidden state (submitted for publication).

Pym, D. J., O'Hearn, P. W. and Yang, H. (2004) Possible worlds and resources: the semantics of BI. *Theoretical Computer Science* **315** (1) 257–305.

Reynolds, J. C. (2002) Separation logic: A logic for shared mutable data structures. In: *Proceedings of LICS* 55–74.

Schwinghammer, J., Birkedal, L., Reus, B. and Yang, H. (2009) Nested Hoare triples and frame rules for higher-order store. In: *Proceedings of CSL* 440–454.

Schwinghammer, J., Birkedal, L. and Støvring, K. (2011) A step-indexed Kripke model of hidden state via recursive properties on recursively defined metric spaces. In: *Proceedings of FOSSACS* 305–319.

Schwinghammer, J., Yang, H., Birkedal, L., Pottier, F. and Reus, B. (2010) A semantic foundation for hidden state. In: *Proceedings of FOSSACS* 2–16.

Smith, F., Walker, D. and Morrisett, G. (2000) Alias types. In: Proceedings of ESOP. *Springer-Verlag Lecture Notes in Computer Science* **1782** 366–381.

Smyth, M. B. (1992) Topology. In: *Handbook of Logic in Computer Science*, volume 1, Oxford University Press.

Wright, A. K. (1995) Simple imperative polymorphism. *Lisp and Symbolic Computation* **8** (4) 343–356.