Step-Indexed Kripke Models over Recursive Worlds

Lars Birkedal
IT University of Copenhagen, Denmark
birkedal@itu.dk

Bernhard Reus
University of Sussex, UK
bernhard@sussex.ac.uk

Jan Schwinghammer
Saarland University, Germany
jan@ps.uni-saarland.de

Kristian Støvring *
University of Copenhagen, Denmark
stovring@diku.dk

Jacob Thamsborg
IT University of Copenhagen, Denmark
thamsborg@itu.dk

Hongseok Yang
Queen Mary University London, UK
hyang@dcs.qmul.ac.uk

Abstract

Over the last decade, there has been extensive research on modelling challenging features in programming languages and program logics, such as higher-order store and storable resource invariants. A recent line of work has identified a common solution to some of these challenges: Kripke models over worlds that are recursively defined in a category of metric spaces. In this paper, we broaden the scope of this technique from the original domain-theoretic setting to an elementary, operational one based on step indexing. The resulting method is widely applicable and leads to simple, succinct models of complicated language features, as we demonstrate in our semantics of Charguéraud and Pottier's type-and-capability system for an ML-like higher-order language. Moreover, the method provides a high-level understanding of the essence of recent approaches based on step indexing.

Categories and Subject Descriptors D.3.1 [*PROGRAMMING LANGUAGES*]: Formal Definitions and Theory; F.3.1 [*LOGICS AND MEANINGS OF PROGRAMS*]: Specifying and Verifying and Reasoning about Programs

General Terms Languages, Theory, Verification

Keywords Kripke models, ultrametric spaces, step-indexed models, capability calculus, frame rules, indirection theory

1. Introduction

Over the last decade, there has been extensive research on modelling challenging features in programming languages, type systems and program logics, such as higher-order store and storable resource invariants, where modelling involves constructing recursively defined structures [15, 22, 29, 32, 41, 42]. One of the main aims of this research has been to develop a method for building semantic models such that (1) the method is simple enough to be understood by the designers of a type system or a program logic (who might have only limited knowledge of domain theory) but (2) the method is powerful enough to resolve the issue of constructing recursive structures.

Unfortunately, existing methods do not fully achieve this aim. Methods based on classical domain theory provide techniques for constructing recursive structures, but they require non-trivial mathematical knowledge from users. Methods based on step indexing [2, 4, 6, 11, 12], on the other hand, do not require sophisticated mathematics from the users; usually, the prerequisite is just familiarity with standard operational semantics of programs. However, the step-indexed methods only partially address the issue of constructing recursive structures. They change the original recursive equations that solutions have to satisfy to easier approximate ones, and construct structures that satisfy the approximate equations. We point out that solving the original recursive equations seems crucial in some applications, such as the semantics of various higher-order frame and anti-frame rules [44, 46]. Hence, in those applications, only domain-theoretic models, not step-indexed ones, have been developed.

In this paper, we propose a new method that brings together the benefits of both domain-theoretic and step-indexing methods. Our approach is based on a recent line of work where challenging features of programming languages and logics are modelled using a common solution: Kripke models over worlds that are recursively defined in a category of metric spaces [21, 44, 46]. This method transfers those worlds from the original domain-theoretic setup to an elementary, operational one based on step indexing.

Although our method does involve a modicum of metric space theory, it retains the flavour and simplicity of traditional step-indexed methods [2, 4, 6, 11, 12]. Unlike these step-indexed models, which only provide solutions to approximated versions of recursive equations, our approach provides solutions to the equations proper, i.e., we solve the equation up to isomorphism. In the paper, we demonstrate the benefits of our method by presenting the first semantic model of Charguéraud and Pottier's capability calculus [24]. This calculus is a substructural type system for a higher-order ML-like language with state. Our semantics justifies the typing rules of the calculus, and it also suggests a sound extension of the type system with a higher-order (deep) frame rule. The latter imposes a nontrivial soundness issue, because we need a recursively defined operation on a recursively-defined set of worlds to model the invariant-extension operator of higher-order framing. [1]

[*] Work carried out while employed at the IT University of Copenhagen.

[1] We have also used the new method to give an elementary *operational* model for a program logic for reasoning about higher-order store; this yields an alternative soundness proof to the earlier non-trivial domain-theoretic

POPL'11, January 26–28, 2011, Austin, Texas, USA.

Our method also provides a high-level understanding of the essence of step-indexed models. In particular, we show that the method can be specialized to Hobor et al.'s recent abstract description of step-indexed models [30], and explain the benefits of taking the metric viewpoint we suggest.

The remainder of the paper is organized as follows. In Section 2, we give an extensive introduction of our method, by developing a step-indexed Kripke model for ML references. In Section 3, we address the challenging problem of modelling Charguéraud and Pottier's capability type system, and show how our method gives rise to a step-indexed Kripke model of the calculus. Next, in Section 4, we consider the connection with the indirection theory of Hobor et al. [30], and point out what new insights our method brings to the work on step indexing. Finally, in Sections 5 and 6, we discuss related work and conclude the paper.

For space reasons, some proofs and details have been omitted. These can be found in the appendices of the long version of the paper, which is available at:

```
http://www.itu.dk/people/birkedal/longessence.pdf
```

2. Introductory Example: ML References

In this section, we give an extensive introduction of our method, using a programming language with impredicative polymorphism and general ML-like references, i.e., an extension of the call-by-value polymorphic lambda calculus with higher-order store. We do not give the syntax of this language as it is standard [36] but point out that we use v for values, e for expressions and τ for types, in particular $e[\tau]$ is application of a polymorphic term to a type.

First, we describe the general idea of interpreting the programming language with a Kripke-style possible-worlds model, where the set of worlds is recursively defined. Then, we review an existing model that realizes the idea in a domain-theoretic setting (based on an adequate denotational semantics of the language). Finally, we present a new step-indexed model (based purely on the operational semantics), and compare it with the domain-theoretic one.

A simple approach for modelling the polymorphic lambda calculus, without general references, is to interpret types as predicates (subsets) on some fixed set of values. To model the programming language of interest now, however, we need to extend this approach, because the language includes dynamic allocation of general references. Following earlier work on the semantics of dynamic allocation of simple integer cells [14, 32, 40, 48], we use an extension with Kripke-style possible worlds. In this extension, a type is interpreted as a predicate on values parameterized over worlds, and a world describes the type for each allocated location—a world $w \in W$ is a finite map from locations (modelled as natural numbers) to semantic types in T. The extension is described by the following recursive equations on the set W of worlds and the set T of semantic types:

$$\begin{aligned} & V \;=\; \text{set of values, including locations} \\ W \;=\; \mathbb{N} \rightharpoonup_{fin} T \qquad & T \;=\; W \to_{mon} Pred(V) \end{aligned} \tag{1}$$

Note that in the equation for T, we impose a monotonicity requirement (with respect to an extension ordering of worlds). Intuitively, this requirement means that validity of semantic types is preserved in presence of a growing heap. The formal meaning of the monotonicity will be explained later in Section 2.2.

Once we are given the semantic domains W and T satisfying the above equations, we can interpret types as elements in T. In particular, the meaning of a reference type $\mathsf{ref}\,\tau$ can be defined roughly as

$$(\mathsf{ref}\,\tau)w = \{l \mid w(l) = \tau\},$$

i.e., for a world w, it is the set of locations l such that the semantic type recorded in the world at l is the same as τ. [2]

Observe that the natural model of types here is a *Kripke model over a recursively-defined set of worlds*. It is a Kripke model because the semantic types are parameterized over W. The problem is, of course, that for cardinality reasons there are no solutions to the above equations in the category of sets; unfolding the above equations we get $W = \mathbb{N} \rightharpoonup_{fin} (W \to_{mon} Pred(V))$ with W in a negative position, see also Ahmed [2].

To address this cardinality issue, existing methods based on step indexing, including the recent work by Hobor et al. [30], propose that we should give up solving the original recursive equations and instead solve approximate versions. As Hobor et al. show, solutions of the approximate equations are often sufficient for the applications of interest.

In this paper, we follow a different approach, which involves finding an appropriate simple category of metric spaces and solving the original recursive equations in the category. This approach has been developed in the setting of denotational semantics and domain theory. We show that the same approach can also be applied to operational semantics and step indexing. In the next subsections, we explain this point by giving a Kripke model of ML references, first using domain-theoretic methods, and then step indexing.

2.1 Review of Metric Spaces

Before describing our Kripke models, we review basic facts on the metric spaces, which will be used in the models; see, e.g., [47] for a thorough exposition. 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 $d(x, y) \leq \max\{d(x, z), d(z, y)\}$. An (ultra)metric space is **complete** if every Cauchy sequence has a limit. A function $f : X_1 \to X_2$ between metric spaces (X_1, d_1) and (X_2, d_2) is **non-expansive** if $d_2(f(x), f(y)) \leq d_1(x, y)$ for all $x, y \in X_1$. It is **contractive** if there exists some $\delta < 1$ such that $d_2(f(x), f(y)) \leq \delta \cdot d_1(x, y)$ for all $x, y \in X_1$.

The complete, 1-bounded, non-empty, ultrametric spaces and non-expansive functions between them form a Cartesian closed category $\mathbf{CBUlt}_{\mathrm{ne}}$. 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 distance function: $d_{X_1 \to X_2}(f, g) = \sup\{d_2(f(x), g(x)) \mid x \in X_1\}$. For any set S and space $(X, d) \in \mathbf{CBUlt}_{\mathrm{ne}}$, the set of finite partial functions $S \rightharpoonup_{fin} X$ from S to X is again a complete, 1-bounded ultrametric space with distance function given by $d(f, g) = 1$, if the domains of f and g are not equal, and $d(f, g) = \max\{d(f(s), g(s)) \mid s \in \mathrm{dom}(f)\}$, if the domains of f and g are equal.

A functor $F : \mathbf{CBUlt}_{\mathrm{ne}}^{\mathrm{op}} \times \mathbf{CBUlt}_{\mathrm{ne}} \longrightarrow \mathbf{CBUlt}_{\mathrm{ne}}$ is **locally non-expansive** if $d(F(f, g), F(f', g')) \leq \max\{d(f, f'), d(g, g')\}$ for all non-expansive f, f', g, g'. It is **locally contractive** if there exists $\delta < 1$ such that $d(F(f, g), F(f', g')) \leq \delta \cdot \max\{d(f, f'), d(g, g')\}$ for all non-expansive f, f', g, g'. By multiplication of the distances of (X, d) with a non-negative shrinking factor $\delta < 1$, one obtains a new ultrametric space, $\delta \cdot (X, d) = (X, d')$ where $d'(x, y) =$

one [44]. Please see the appendix of the long version of the paper for this model.

[2] In both of the concrete models to be presented next, the interpretation of reference types is actually more complicated and involves certain "approximate equality" relations on semantic types. This is fairly standard for step-indexed models; that it is also required in the domain-theoretic setup is discussed by Birkedal et al. [21].

$\delta \cdot d(x, y)$. By shrinking, a locally non-expansive functor F yields a locally contractive functor $(\delta \cdot F)(X_1, X_2) = \delta \cdot (F(X_1, X_2))$.

It is well-known that one can solve recursive domain equations in $\mathbf{CBUlt}_{\mathrm{ne}}$ by an adaptation of the inverse-limit method from classical domain theory:

Theorem 2.1 (America-Rutten [9])**.** Let $F : \mathbf{CBUlt}_{\mathrm{ne}}^{\mathrm{op}} \times \mathbf{CBUlt}_{\mathrm{ne}} \to \mathbf{CBUlt}_{\mathrm{ne}}$ be a locally contractive functor. Then there exists a unique (up to isomorphism) $(X, d) \in \mathbf{CBUlt}_{\mathrm{ne}}$ such that $(X, d) \cong F((X, d), (X, d))$.

All the metric spaces we consider satisfy the following property:

Definition 2.2. A metric space is *bisected* if all non-zero distances are of the form 2^{-n} for some natural number $n \geq 0$.

The following notation is convenient when working with bisected metric spaces: in such a space, $x \overset{n}{=} y$ means that $d(x, y) \leq 2^{-n}$. We use two facts on $\overset{n}{=}$. First, each relation $\overset{n}{=}$ is an equivalence relation because of the ultrametric inequality. We are therefore justified in referring to the relation $\overset{n}{=}$ as "n-equality." Second, the distance of a bisected metric space is bounded by 1. In other words, the relation $x \overset{0}{=} y$ always holds.

Proposition 2.3. Let (X_1, d_1) and (X_2, d_2) be bisected metric spaces. A function $f : X_1 \to X_2$ is non-expansive if and only if $x_1 \overset{n}{=} x_1' \Rightarrow f(x_1) \overset{n}{=} f(x_1')$ holds for all $x_1, x_1' \in X_1$ and all natural numbers $n \geq 0$.

2.2 General Recipe and Domain-Theoretic Model

We now follow the idea outlined earlier and reformulate the recursive equations (1) in $\mathbf{CBUlt}_{\mathrm{ne}}$ to find solutions within this category. Concretely, the proposal suggests to use the recipe below:

1. Define a set V with a structure. The structure can be a preorder, or a uniform complete partial order, but does not have to be. Intuitively, V is a domain for semantic values.
2. Define an object $Pred(V)$ in $\mathbf{CBUlt}_{\mathrm{ne}}$. Elements in this object represent predicates on values.
3. Solve the recursive domain equation below in $\mathbf{CBUlt}_{\mathrm{ne}}$:
$$\hat{T} \;\cong\; \tfrac{1}{2} \cdot ((\mathbb{N} \rightharpoonup_{fin} \hat{T}) \to_{mon} Pred(V))\,. \tag{2}$$
4. Define T and W using $\hat{T}$:
$$W \;=\; N \rightharpoonup_{fin} \hat{T}, \qquad T \;=\; W \to_{mon} Pred(V). \tag{3}$$

The function space in the equivalence in the third step consists of non-expansive and monotone functions, where monotonicity is imposed with respect to the following extension order on $\mathbb{N} \rightharpoonup_{fin} T$: For $w, w' \in \mathbb{N} \rightharpoonup_{fin} T$, we have $w \sqsubseteq w'$ iff the domain of w is included in the domain of w', and w and w' agree on the former. The $\frac{1}{2}$ is an example of a shrinking factor and, technically, ensures that the functor is locally contractive; it is a standard technique [9]. The equivalence is well-formed in $\mathbf{CBUlt}_{\mathrm{ne}}$, and it has a unique solution up to isomorphism by Theorem 2.1.

The recipe has been used by Birkedal, Støvring and Thamsborg [21], when they gave a relationally-parametric domain-theoretic model of a call-by-value language with impredicative polymorphism, general references and recursive types. They constructed the parameters V and $Pred(V)$ of the recipe using domain theory, choosing for V the cpo of values that is used in the standard "untyped" domain-theoretic interpretation of the language. This domain V comes with a family of projections $\pi_n : V \to V_\bot$ satisfying certain properties (so it becomes a uniform cpo). For the next parameter $Pred(V)$, Birkedal et al. used these projections to define $Pred(V)$ as the collection of *complete uniform* subsets of V. *Completeness* says that a subset P is closed under least upper bounds of chains, and *uniformity* that P is closed under all the projections (i.e., $\forall v \in P.\ \forall n > 0.\ \pi_n(v) \in P_\bot$). The set $Pred(V)$ can be viewed as a metric space in $\mathbf{CBUlt}_{\mathrm{ne}}$, by giving it an appropriate distance function along the lines of earlier work on interpreting recursive types and impredicative polymorphism [1, 7, 8, 23, 33].

Now, by simply following the recipe from the given ingredients (i.e. parameters V and $Pred(V)$) one obtains metric spaces T for semantic types and W for possible worlds, respectively. With this indexed semantic model of types, Birkedal et al. gave an interpretation of all the types of the programming language, and defined the typed meaning of expressions by proving the fundamental theorem of logical relations wrt. the untyped semantics of expressions. See [21] for a detailed treatment.[3]

2.3 Step-Indexed Model

Our new insight is that the recipe presented in Section 2.2 is not tied to domain theory and denotational semantics, but it can also be used with operational semantics. In this case, the first parameter of the recipe is the set *Val* of closed syntactic values from the operational semantics. The second parameter is the set of predicates on step-approximated values. Precisely, it is the collection $UPred(\mathit{Val})$ of subsets of $\mathbb{N} \times \mathit{Val}$ that are downwards closed in the first step $(\mathbb{N})$ component:

$$UPred(\mathit{Val}) \;=\; \{p \subseteq \mathbb{N} \times \mathit{Val} \mid \forall (k, v) \in p.\, \forall j \leq k.\, (j, v) \in p\}.$$

We call $p \in UPred(\mathit{Val})$ a **uniform predicate** on *Val*.

Note that we give a unary interpretation; in particular the model does not prove any program equivalences. This is just for simplicity of presentation, a relational, i.e., binary, model can be built along the same lines using $UPred(\mathit{Val} \times \mathit{Val})$.

The idea of considering predicates on step-approximated values is from step-indexed models [2, 4, 11, 12]. Here we go "a step further" and show that the collection $UPred(\mathit{Val})$ of such predicates can always be made into an object in $\mathbf{CBUlt}_{\mathrm{ne}}$. To do this, for $p \in UPred(\mathit{Val})$ and $k \in \mathbb{N}$, we use the notation $p_{[k]} = \{(m, v) \in p \mid m < k\}$, representing the k-th approximation of p. With this notation, we define a distance function d on $UPred(\mathit{Val})$, which measures "up-to-what-level" two predicates agree:

$$d(p, q) \;=\; \begin{cases} 2^{-\max\{k \mid p_{[k]} = q_{[k]}\}} & \text{if } p \neq q \\ 0 & \text{otherwise.} \end{cases}$$

Lemma 2.4. $(UPred(\mathit{Val}), d)$ is a well-defined object in $\mathbf{CBUlt}_{\mathrm{ne}}$. In fact, the construction in $UPred(\mathit{Val})$ does not depend on our choice of *Val*, and can be applied to any set X, giving a metric space $UPred(X)$ in $\mathbf{CBUlt}_{\mathrm{ne}}$.

Note that because of this lemma, we can consider uniform predicates $p \in UPred(X)$ on any set X.

Hence, the recipe in Section 2.2 is applicable for *Val* and $UPred(\mathit{Val})$, and gives rise to semantic domains $\hat{T}$, W and T that satisfy the recursive equations in (2) and (3). Note that by working in $\mathbf{CBUlt}_{\mathrm{ne}}$, we have solved the desired equations, even for a setting based on operational semantics. In the rest of this section, we use these domains and model the programming language with impredicative polymorphism and ML references.

For concreteness, we consider a language as in Dreyer et al. [27], except that we do not consider recursive types and we split the context for type variables and term variables in two. Term

[3] Notice that we use a small "trick" to construct the space of worlds W using Theorem 2.1. By solving the equation (2) we first obtain the space of semantic types, and we then define worlds in terms of semantic types. It is also possible to obtain W directly, as a solution of a recursive equation in a category of *pre-ordered* ultrametric spaces [20]. The latter technique is more general, but for this paper we do not need such pre-ordered spaces.

$$\llbracket \Xi \vdash \tau \rrbracket_\eta : W \to_{mon} \mathit{UPred}(\mathit{Val})$$

$$\begin{aligned}
\llbracket \Xi \vdash 1 \rrbracket_\eta w &= \{(k, ()) \mid k \in \mathbb{N}\} \\
\llbracket \Xi \vdash \mathsf{ref}\,\tau \rrbracket_\eta w &= \{(k, l) \mid l \in \mathrm{dom}(w) \wedge w(l) \overset{k}{=} \llbracket \Xi \vdash \tau \rrbracket_\eta\} \\
\llbracket \Xi \vdash \alpha \rrbracket_\eta w &= \eta(\alpha)(w) \\
\llbracket \Xi \vdash \forall\alpha.\tau \rrbracket_\eta w &= \{(k, v) \mid \forall\tau' \in \mathit{SyntacticType}.\ \forall r \in T.\ \forall w' \sqsupseteq w. \\
&\qquad \forall i \leq k.\ (i, v[\tau']) \in \mathcal{E}\llbracket \Xi, \alpha \vdash \tau \rrbracket_{\eta[\alpha \mapsto r]} w'\} \\
\llbracket \Xi \vdash \tau \to \tau' \rrbracket_\eta w &= \{(k, v) \mid \forall v' \in \mathit{Val}.\ \forall w' \sqsupseteq w.\ \forall i \leq k. \\
&\qquad (i, v') \in \llbracket \Xi \vdash \tau \rrbracket_\eta w' \Rightarrow (i, v\,v') \in \mathcal{E}\llbracket \Xi \vdash \tau' \rrbracket_\eta w'\}
\end{aligned}$$

$$\begin{aligned}
\mathcal{E}\llbracket \Xi \vdash \tau \rrbracket_\eta w = \{(k, t) \mid \forall i \leq k.\ \forall h, h'.\ \forall e'.\ \big(&h :_k w \wedge \\
&(t \mid h) \longmapsto^i (e' \mid h') \wedge (e', h') \text{ irreducible}\big) \Rightarrow \\
&\big(\exists w' \sqsupseteq w.\ \ h' :_{k-i} w' \ \wedge\ (k-i, e') \in \llbracket \Xi \vdash \tau \rrbracket_\eta w'\big)\}
\end{aligned}$$

$$\begin{aligned}
h :_k w \iff\ & \mathrm{dom}(h) = \mathrm{dom}(w) \ \wedge \\
& \forall i < k.\,\forall l \in \mathrm{dom}(w).\ \ (i, h(l)) \in w(l)(w)
\end{aligned}$$

(where $()$ is the unique element in the empty product, $\mathit{SyntacticType}$ is the set of syntactic types, *Exp* is the set of closed syntactic expressions, and $(t \mid h)$ is a configuration of an expression t and a heap h. A heap h is a finite function from locations to closed syntactic values.)

Figure 1. Interpretation of types

judgments take the form

$$\Xi; \Gamma; \Sigma \ \vdash\ M : \tau$$

where Ξ is a context of type variables $\alpha_1, \ldots, \alpha_n$; Γ is a context of typed term variables $x_1 : \tau_1, \ldots, x_m : \tau_m$; and Σ is a context of typed locations $l_1 : \tau_1, \ldots, l_k : \tau_k$. Detailed typing judgments and operational semantics can be found in the online appendix to Dreyer et al.

Types in this language are interpreted similar to those used in existing step-indexed models [2], but one can exploit the fact that W and T are solutions to the recursive equations above. The semantics of types in context is defined as a non-expansive function

$$\llbracket \Xi \vdash \tau \rrbracket \ :\ T^{|\Xi|} \to T$$

in **CBUlt**$_{\mathrm{ne}}$. The definition is shown in Figure 1. In the figure, we use η for environments for Ξ, i.e., elements in the product space $T^{|\Xi|}$ in **CBUlt**$_{\mathrm{ne}}$. Notice that in the case for $\llbracket \Xi \vdash \mathsf{ref}\,\tau \rrbracket$, we use k-equality in the space T and that $\mathcal{E}\llbracket \Xi \vdash \tau \rrbracket$ generalizes $\llbracket \Xi \vdash \tau \rrbracket$ from values to expressions.

Lemma 2.5. $\llbracket \Xi \vdash \tau \rrbracket$ is well-defined. In particular,

- for all $\eta \in T^{|\Xi|}$, $\llbracket \Xi \vdash \tau \rrbracket_\eta$ is non-expansive and monotone; and
- $\llbracket \Xi \vdash \tau \rrbracket$ is a non-expansive map on η's.

In Figure 2 we define interpretations of contexts and the logical relation interpretation of well-typed expressions. Using those definitions, we are ready to prove the main soundness result:

Theorem 2.6 (Fundamental Theorem of Logical Relations)**.** If $\Xi; \Gamma; \Sigma \vdash t : \tau$, then $\Xi; \Gamma; \Sigma \vdash t :^{\log} \tau$.

One oddity is worth explaining: there is no coherence between the syntactic types that we substitute for type variables and the corresponding semantic types in the environment; this is the case both for the interpretation of universal types and in the definition of the logical relation. The explanation is simply that the syntactic types in values and expressions do not influence the computation; indeed, we could equally well have worked with a language without type-decorations as, e.g., Ahmed [3] does.

$$\llbracket \Xi \vdash \Gamma \rrbracket_\eta : W \to \mathit{UPred}(\mathit{Val}^{|\Gamma|})$$

$$\begin{aligned}
\llbracket \Xi \vdash \emptyset \rrbracket_\eta w &= \{(k, ()) \mid k \in \mathbb{N}\} \\
\llbracket \Xi \vdash \Gamma, x : \tau \rrbracket_\eta w &= \{(k, \rho[x \mapsto v]) \mid (k, \rho) \in \llbracket \Xi \vdash \Gamma \rrbracket_\eta w \ \wedge \\
&\qquad (k, v) \in \llbracket \Xi \vdash \tau \rrbracket_\eta w\}
\end{aligned}$$

$$\llbracket \Sigma \rrbracket : \mathit{UPred}(W)$$

$$\llbracket \Sigma \rrbracket = \{(k, w) \mid \forall (l : \tau) \in \Sigma.\ (k, l) \in \llbracket \emptyset \vdash \mathsf{ref}\,\tau \rrbracket_{()} w\}$$

$$\begin{aligned}
&\Xi; \Gamma; \Sigma \vdash t :^{\log} \tau \iff \\
&\quad \exists \alpha_1, \ldots, \alpha_n.\ \Xi = \alpha_1, \ldots, \alpha_n \ \wedge \\
&\quad \forall \tau_1, \ldots, \tau_n \in \mathit{SyntacticType}.\ \forall k \geq 0.\ \forall \eta.\ \forall \rho.\ \forall w. \\
&\qquad \big(\eta \in T^{|\Xi|} \ \wedge\ (k, \rho) \in \llbracket \Xi \vdash \Gamma \rrbracket_\eta w \ \wedge\ (k, w) \in \llbracket \Sigma \rrbracket\big) \\
&\qquad \Rightarrow \big((k,\ (\rho(t))[\alpha_1{:=}\tau_1, \ldots, \alpha_n{:=}\tau_n]) \in \mathcal{E}\llbracket \Xi \vdash \tau \rrbracket_\eta w\big)
\end{aligned}$$

(where $()$ is the unique environment for the empty context, and both $(-)[\alpha_1{:=}\tau_1, \ldots, \alpha_n{:=}\tau_n]$ and $\rho(-)$ represent the applications of substitutions.)

Figure 2. Interpretation of contexts and well-typed expressions

We finally remark that it is not surprising that there is a connection between metric spaces and step-indexed models; this was already pointed out in [11]. The point is that it is useful not to forget this connection because it, e.g., allows us to define solutions to recursive world equations such as the ones in this section. (See also the discussion in Section 4.2.)

We do not present a formal relationship to existing models for this particular example, but rather show, in Section 4, how all the step-indexed models described via the indirection theory of Hobor et al. [30] can be obtained by a specialization of our general approach. In Section 4.2, we will also highlight the advantages of using metric spaces. Next, however, we consider another more substantial application to illustrate our method.

3. Application: A Step-indexed Model of Capabilities

Reasoning about higher-order stateful programs is notoriously difficult, and often involves the need to track aliasing information. A particular line of work that has been proposed to this end are substructural type systems with regions, capabilities and singleton types [5, 24, 26]. In this section, we give a step-indexed model for a substantial fragment[4] of Charguéraud and Pottier's capability calculus [24]. Our model provides an alternative soundness proof to the translation and progress and preservation results in [24, 39], and allows for the analysis of soundness of extensions. We illustrate this latter point by proving sound an extension of the language with higher-order frame rules [19, 44], and establish an explicit connection with models of separation logic *qua* our model, which shows that capabilities can be understood semantically as separation logic predicates, i.e., as predicates on heaps.

We believe that this step-indexed model provides an interesting application of the metric point of view that has been emphasized in the previous section. The model construction takes advantage of the fact that the recursive world equation can be *solved* (up to isomorphism), rather than merely *approximated*: the higher-order frame rules are modelled with the help of a recursive operation on worlds, and this operation is defined using the metric structure.

[4] We do not consider group regions.

$$\begin{aligned} v &::= x \mid () \mid \mathsf{inj}^i\, v \mid (v_1, v_2) \mid \mu f.\lambda x.t \mid l \\ t &::= v \mid (v\,t) \mid \mathsf{case}(v_1, v_2, v) \mid \mathsf{proj}^i\, v \mid \mathsf{ref}\, v \mid \mathsf{get}\, v \mid \mathsf{set}\, v \end{aligned}$$

Figure 3. Syntax of values and expressions

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

Figure 4. Operational semantics

While we do not model the anti-frame rule [39] here, the approach does scale to do that; this is demonstrated in recent work by Schwinghammer, Birkedal and Støvring [45].

3.1 A Calculus of Capabilities

In the following presentation, we keep close to the notation of Charguéraud and Pottier [24, 39]. Figures 3 and 4 give the syntax and operational semantics of the programming language that we consider. It is a standard call-by-value, higher-order language with general references, and polymorphic and recursive types. The only noteworthy point about the syntax is that expressions are restricted so that all sequencing is made explicit; this simplifies the presentation of the typing rules and semantics but is no real restriction. The term $\mu f.\lambda x.t$ stands for the recursive procedure f with body t and argument x. If f does not appear in t, we may simply write $\lambda x.t$.

The operational semantics is defined between configurations $(t \mid h)$ that consist of a (closed) expression t and a heap h. As in the previous section, a heap h is a finite map from locations to closed values. Also, we remind the reader of our notation $t[x{:=}v]$ that means the substitution of v for x in t. We use the notation $h \# h'$ to indicate that two heaps h and h' have disjoint domains, and we write $h \cdot h'$ for the union of two such heaps.

The types used in the system are given by the grammar in Figure 5. *Capabilities C* describe heap properties (much like the assertions of a Hoare-style program logic), *value types τ* classify values, and *memory types θ* (and the subset of computation types) describe properties of expressions and how their evaluation affects the heap. Because of the heap dependency, capabilities and memory types are linear, and correspondingly there is a distinction between value type environments and the more general linear environments.

A *region* σ is a static name that represents a value, and $[\sigma]$ is a singleton type that contains only this particular value. Capabilities are formed from *singleton capabilities* $\{\sigma : \theta\}$ by separating conjunction and existential quantification over regions. We also include capability variables γ and permit recursively defined capabilities. A singleton capability $\{\sigma : \theta\}$ asserts that the value denoted by σ has type θ, and moreover it represents the ownership of both this value and the fragment of the heap described by θ. Thus, it is similar to the points-to predicate of separation logic: for example, the capability $\{\sigma : \mathsf{ref}\,\tau\}$ means that σ denotes the address of a reference cell, and that the "owned" part of the heap stores a value of type τ at this address. Apart from singleton types, the value types include base types (here an empty type 0, the unit type 1, and int) and are closed under products, sums, and universal quantification over singletons, types and capabilities. The memory types extend value types by a type of references, and by the possibility to $*$-conjoin a capability. Like the pre- and postconditions used in Hoare logic, the arrow types make explicit which part of the heap is accessed when a procedure is called. For instance, the type $\forall \sigma, \sigma'.\, [\sigma] * \{\sigma : \mathsf{ref}\, [\sigma']\} \to [\sigma'] * \{\sigma : \mathsf{ref}\, [\sigma']\}$ can be given to a procedure that dereferences its argument.

Recursive capabilities and types are subject to a syntactic restriction: C must be *formally contractive* in γ for $\mu\gamma.C$ to be well-formed. By this we mean that the recursion must go through one of the type constructors $+$, $\times$, $\to$ or ref, or through the right-hand side of $\otimes$. This restriction ensures that the capability $\mu\gamma.C$ is the unique solution of the capability equation $\gamma = C$. Corresponding restrictions apply to recursively defined types $\mu\alpha.\tau$ and $\mu\beta.\theta$. We omit the straightforward inductive definition of formal contractiveness.

One interesting aspect of the type system is that each of the 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 functions of this type. This intuition is made precise by regarding capabilities and types modulo the structural equivalence given in Figure 6. This equivalence subsumes the "distribution axioms" for $\otimes$ that are used to express generic higher-order frame rules [19]. The first two groups of equations, equivalences (4)–(11), state that both $*$ and the derived operation $\circ$ on capabilities satisfy the axioms of a monoid, and that $*$ and $\otimes$ are actions of these monoids. Equivalences (15)–(30) describe the action by $\otimes$ on types. In particular, (25) shows the key case of the invariant extension described informally above.[5] Finally, the equivalences (34)–(38) for *focusing* let us build and deconstruct the capabilities over complex types in terms of capabilities over more primitive types.

The system also uses a subtyping relation, and Figure 7 gives some of the subtyping axioms. The typing rules are shown in Figure 8. Due to the use of linear environments and computation types (which in general contain embedded capabilities), the typing judgement $\Gamma \Vdash t : \chi$ is similar to a Hoare triple where Γ serves as a precondition and χ as a postcondition. This view explains the rules SHALLOW-FRAME and DEEP-FRAME; as in separation logic, these rules can be used to add an invariant C to a specification. The difference between SHALLOW-FRAME and DEEP-FRAME is that the former adds C only on the top-level, whereas the latter also extends all arrow types nested inside Γ and χ, via $\cdot \otimes C$. As with the higher-order frame rules in separation logic, this is useful for reasoning about information hiding [19].

3.2 Upwards Closed Uniform Predicates and Worlds

The main idea of the model that we present next is that types (as well as type contexts and capabilities) are parameterized by invariants. Thus, in this case the worlds will be predicates that, like the syntactic capabilities of the calculus, describe properties of the heap that all computations must preserve.

Recall that the set $\mathit{UPred}(X)$ of uniform predicates on a set X is defined by

$$\mathit{UPred}(X) \;=\; \{p \subseteq \mathbb{N} \times X \mid \forall (k, v) \in p.\, \forall j \leq k.\, (j, v) \in p\}.$$

The interpretation of types and capabilities is based on a variation on these uniform predicates. Let $(A, \sqsubseteq)$ be a partially ordered set. An *upwards closed* uniform predicate p on A is a predicate in $\mathit{UPred}(A)$ that is also upward closed in the second argument, i.e. if $(k, a) \in p$ and $a \sqsubseteq b$ then $(k, b) \in p$. We write $\mathit{UPred}^{\uparrow}(A)$ for

[5] Note that (13) and (14) let us move capabilities between assumptions – a form of ownership transfer.

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	$\tau ::= \tau \otimes C \mid 0 \mid 1 \mid \mathsf{int} \mid \tau + \tau \mid \tau \times \tau \mid \chi \rightarrow \chi \mid [\sigma] \mid \alpha \mid \mu\alpha.\tau \mid \forall\xi.\tau$
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 environments	$\Delta ::= \Delta \otimes C \mid \varnothing \mid \Delta, x{:}\tau$
Linear environments	$\Gamma ::= \Gamma \otimes C \mid \varnothing \mid \Gamma, x{:}\chi \mid \Gamma * C$

Figure 5. Capabilities and types

$$\textsc{Var}\ \frac{(x:\tau) \in \Delta}{\Delta \vdash x : \tau} \qquad \textsc{Unit}\ \frac{}{\Delta \vdash () : 1} \qquad \textsc{Inj}\ \frac{\Delta \vdash v : \tau_i}{\Delta \vdash (\mathsf{inj}^i\ v) : (\tau_1{+}\tau_2)} \qquad \textsc{Pair}\ \frac{\Delta \vdash v_1 : \tau_1 \qquad \Delta \vdash v_2 : \tau_2}{\Delta \vdash (v_1, v_2) : (\tau_1 \times \tau_2)} \qquad \textsc{RecFun}\ \frac{\Delta,\, f : \chi_1 \rightarrow \chi_2,\, x : \chi_1 \Vdash t : \chi_2}{\Delta \vdash \mu f.\lambda x.t : \chi_1 \rightarrow \chi_2}$$

$$\textsc{Val}\ \frac{\Delta \vdash v : \tau}{\Delta \Vdash v : \tau} \qquad \textsc{App}\ \frac{\Delta \vdash v : \chi_1 \rightarrow \chi_2 \qquad \Delta, \Gamma \Vdash t : \chi_1}{\Delta, \Gamma \Vdash (v\,t) : \chi_2} \qquad \textsc{Proj-1}\ \frac{\Gamma \Vdash v : [\sigma] * \{\sigma : \tau_1 \times \theta_2\}}{\Gamma \Vdash \mathsf{proj}^1\, v : \tau_1 * \{\sigma : \tau_1 \times \theta_2\}} \qquad \textsc{Proj-2}\ \frac{\Gamma \Vdash v : [\sigma] * \{\sigma : \theta_1 \times \tau_2\}}{\Gamma \Vdash \mathsf{proj}^2\, v : \tau_2 * \{\sigma : \theta_1 \times \tau_2\}}$$

$$\textsc{Case}\ \frac{\begin{array}{c}\Delta \vdash v_1 : (\exists\sigma_1.[\sigma_1] * \{\sigma : [\sigma_1] + 0\} * \{\sigma_1 : \theta_1\} * C) \rightarrow \chi \\ \Delta \vdash v_2 : (\exists\sigma_2.[\sigma_2] * \{\sigma : 0 + [\sigma_2]\} * \{\sigma_2 : \theta_2\} * C) \rightarrow \chi \\ \Delta, \Gamma \Vdash v : [\sigma] * \{\sigma : \theta_1 + \theta_2\} * C\end{array}}{\Delta, \Gamma \Vdash \mathsf{case}(v_1, v_2, v) : \chi} \qquad \forall\textsc{-Intro}\ \frac{\Delta \vdash v : \tau}{\Delta \vdash v : \forall\xi.\tau}\,\xi \notin \Delta \qquad \forall\textsc{-Elim-1}\ \frac{\Delta \vdash v : \forall\alpha.\tau}{\Delta \vdash v : \tau[\alpha := \tau']}$$

$$\textsc{Ref}\ \frac{\Gamma \Vdash v : \tau}{\Gamma \Vdash \mathsf{ref}\, v : \exists\sigma.[\sigma] * \{\sigma : \mathsf{ref}\,\tau\}} \qquad \textsc{Get}\ \frac{\Gamma \Vdash v : [\sigma] * \{\sigma : \mathsf{ref}\,\tau\}}{\Gamma \Vdash \mathsf{get}\, v : \tau * \{\sigma : \mathsf{ref}\,\tau\}} \qquad \textsc{Set}\ \frac{\Gamma \Vdash v : ([\sigma] \times \tau_2) * \{\sigma : \mathsf{ref}\,\tau_1\}}{\Gamma \Vdash \mathsf{set}\, v : 1 * \{\sigma : \mathsf{ref}\,\tau_2\}}$$

$$\textsc{Shallow-Frame}\ \frac{\Gamma \Vdash t : \chi}{\Gamma * C \Vdash t : \chi * C} \qquad \textsc{Deep-Frame}\ \frac{\Gamma \Vdash t : \chi}{(\Gamma \otimes C) * C \Vdash t : (\chi \otimes C) * C} \qquad \textsc{Sub}\ \frac{\Gamma \Vdash t : \chi_1 \qquad \chi_1 \leq \chi_2}{\Gamma \Vdash t : \chi_2}$$

Figure 8. Typing of values and expressions

the set of all upwards closed uniform predicates on A, and define

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

As in Section 2.3 on $UPred(V)$, this restricts p to pairs with first component less than k. Note that $p_{[k]}$ is again upwards closed and uniform, so it belongs to $UPred^{\uparrow}(A)$ as well. We equip $UPred^{\uparrow}(A)$ with the same distance function d as $UPred(A)$ in Section 2.3. This makes $(UPred^{\uparrow}(A), d)$ an object of $\mathbf{CBUlt}_{\mathrm{ne}}$.

In our model, we use $UPred^{\uparrow}(A)$ with the following concrete instances for the partial order $(A, \sqsubseteq)$:

- $(Heap, \sqsubseteq)$ where $h \sqsubseteq h'$ iff $h' = h \cdot h_0$ for some $h_0 \# h$,
- $(Val, \sqsubseteq)$ where $u \sqsubseteq v$ iff $u = v$,
- $(Val \times Heap, \sqsubseteq)$ where $(u, h) \sqsubseteq (v, h')$ iff $u = v$ and $h \sqsubseteq h'$.

As in Section 2.3, *Val* is the set of closed values of the language, see Figure 3. We also use variants of the latter two instances where the set *Val* is replaced by the set of value substitutions, *Env*, and by the set of expressions, *Exp*.

On $UPred^{\uparrow}(Heap)$, ordered by subset inclusion, we have a complete Heyting BI algebra structure [17]. Meets and joins are given by set-theoretic intersections and unions, resp., and implication, separating conjunction and separating implication are given by

$$\begin{aligned} (k, h) \in p \rightarrow q \;&\Leftrightarrow\; \forall j \leq k.\ \forall h' \sqsupseteq h.\ (j, h') \in p \Rightarrow (j, h') \in q \\ (k, h) \in p_1 * p_2 \;&\Leftrightarrow\; \exists h_1, h_2.\ h = h_1{\cdot}h_2 \,\wedge\, (k, h_i) \in p_i \\ (k, h) \in p \mathrel{-\!\!*} q \;&\Leftrightarrow\; \forall j \leq k.\ \forall h' \# h.\ (j, h') \in p \Rightarrow (j, h{\cdot}h') \in q \end{aligned}$$

The unit for $*$ is given by $I \;=\; \mathbb{N} \times Heap \;=\; \top$. Up to the natural number indexing, this is just the standard intuitionistic (in the sense that it is not "tight") model of separation logic [43].

Since the worlds are to represent invariants (for instance, describing the shape of data structures laid out in the heap) and since the language of Section 3.1 has general references (so these invariants talk about stored procedures and are themselves world-dependent), it is natural that worlds $w \in W$ must also double-act as functions $W \rightarrow UPred^{\uparrow}(Heap)$. Consequently, we solve in $\mathbf{CBUlt}_{\mathrm{ne}}$ the following recursive world equation:

$$W \;\;\cong\;\; \tfrac{1}{2} \cdot W \;\rightarrow\; UPred^{\uparrow}(Heap)\,. \tag{46}$$

Here, the function space is that of $\mathbf{CBUlt}_{\mathrm{ne}}$ and the $1/2$ denotes the scaling of the distance function on W. That W exists (and is uniquely determined up to isomorphism) follows from Theorem 2.1, applied to the locally contractive functor $F(X, Y) = \frac{1}{2} \cdot X \rightarrow UPred^{\uparrow}(Heap)$ on $\mathbf{CBUlt}_{\mathrm{ne}}$. Worlds are thus essentially *contractive* functions from worlds to $UPred^{\uparrow}(Heap)$, i.e. world dependent heap predicates. We define

$$Cap \;\;=\;\; \tfrac{1}{2} \cdot W \rightarrow UPred^{\uparrow}(Heap),$$

monoids

$$C_1 \circ C_2 \overset{def}{=} (C_1 \otimes C_2) * C_2 \qquad (4)$$

$$(C_1 \circ C_2) \circ C_3 = C_1 \circ (C_2 \circ C_3) \qquad (5)$$

$$C \circ \emptyset = C \qquad (6)$$

$$(C_1 * C_2) * C_3 = C_1 * (C_2 * C_3) \qquad (7)$$

$$C * \emptyset = C \qquad (8)$$

$$C_1 * C_2 = C_2 * C_1 \qquad (9)$$

monoid actions

$$(\cdot \otimes C_1) \otimes C_2 = \cdot \otimes (C_1 \circ C_2) \qquad \cdot \otimes \emptyset = \cdot \qquad (10)$$

$$(\cdot * C_1) * C_2 = \cdot * (C_1 * C_2) \qquad \cdot * \emptyset = \cdot \qquad (11)$$

action by $*$ on singleton

$$\{\sigma : \theta\} * C = \{\sigma : \theta * C\} \qquad (12)$$

action by $*$ on linear environments

$$(\Gamma, x{:}\chi) * C = \Gamma, x{:}(\chi * C) \qquad (13)$$

$$= (\Gamma * C), x{:}\chi \qquad (14)$$

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

$$(\cdot * \cdot) \otimes C = (\cdot \otimes C) * (\cdot \otimes C) \qquad (15)$$

$$(\exists \sigma . \cdot) \otimes C = \exists \sigma . (\cdot \otimes C) \qquad \text{if } \sigma \notin \mathit{RegNames}(C) \qquad (16)$$

$$\emptyset \otimes C = \emptyset \qquad (17)$$

$$\{\sigma : \theta\} \otimes C = \{\sigma : \theta \otimes C\} \qquad (18)$$

$$0 \otimes C = 0 \qquad (19)$$

$$1 \otimes C = 1 \qquad (20)$$

$$\mathsf{int} \otimes C = \mathsf{int} \qquad (21)$$

$$(\theta_1 + \theta_2) \otimes C = (\theta_1 \otimes C) + (\theta_2 \otimes C) \qquad (22)$$

$$(\theta_1 \times \theta_2) \otimes C = (\theta_1 \otimes C) \times (\theta_2 \otimes C) \qquad (23)$$

$$(\forall \xi . \theta) \otimes C = \forall \xi . (\theta \otimes C) \qquad \text{if } \xi \notin \mathrm{fv}\, C \qquad (24)$$

$$(\chi_1 \to \chi_2) \otimes C = (\chi_1 \circ C) \to (\chi_2 \circ C) \qquad (25)$$

$$[\sigma] \otimes C = [\sigma] \qquad (26)$$

$$(\mathsf{ref}\, \theta) \otimes C = \mathsf{ref}\, (\theta \otimes C) \qquad (27)$$

$$\varnothing \otimes C = \varnothing \qquad (28)$$

$$(\Gamma, x{:}\chi) \otimes C = (\Gamma \otimes C), x{:}(\chi \otimes C) \qquad (29)$$

$$(\Gamma * C_1) \otimes C_2 = (\Gamma \otimes C_2) * (C_1 \otimes C_2) \qquad (30)$$

region abstraction

$$\exists \sigma_1 . \exists \sigma_2 . \cdot = \exists \sigma_2 . \exists \sigma_1 . \cdot \qquad (31)$$

$$\cdot * (\exists \sigma . C) = \exists \sigma . (\cdot * C) \qquad (32)$$

$$\{\sigma_1 : \exists \sigma_2 . \theta\} = \exists \sigma_2 . \{\sigma_1 : \theta\} \qquad \text{where } \sigma_1 \neq \sigma_2 \qquad (33)$$

focusing

$$\{\sigma_1 : \mathsf{ref}\, \theta\} = \exists \sigma_2 . \{\sigma_1 : \mathsf{ref}\, [\sigma_2]\} * \{\sigma_2 : \theta\} \qquad (34)$$

$$\{\sigma : \theta_1 \times \theta_2\} = \exists \sigma_1 . \{\sigma : [\sigma_1] \times \theta_2\} * \{\sigma_1 : \theta_1\} \qquad (35)$$

$$\{\sigma : \theta_1 \times \theta_2\} = \exists \sigma_2 . \{\sigma : \theta_1 \times [\sigma_2]\} * \{\sigma_2 : \theta_2\} \qquad (36)$$

$$\{\sigma : \theta_1 + 0\} = \exists \sigma_1 . \{\sigma : [\sigma_1] + 0\} * \{\sigma_1 : \theta_1\} \qquad (37)$$

$$\{\sigma : 0 + \theta_2\} = \exists \sigma_2 . \{\sigma : 0 + [\sigma_2]\} * \{\sigma_2 : \theta_2\} \qquad (38)$$

recursion

$$\mu \gamma . C = C[\gamma{:=}\mu \gamma . C] \qquad (39)$$

$$\mu \alpha . \tau = \tau[\alpha{:=}\mu \alpha . \tau] \qquad (40)$$

$$\mu \beta . \theta = \theta[\beta{:=}\mu \beta . \theta] \qquad (41)$$

Figure 6. Structural equivalence

(first-order) frame axiom

$$\chi_1 \to \chi_2 \leq (\chi_1 * C) \to (\chi_2 * C) \qquad (42)$$

free

$$C_1 * C_2 \leq C_1 \qquad (43)$$

singletons

$$\tau \leq \exists \sigma . [\sigma] * \{\sigma : \tau\} \qquad (44)$$

$$[\sigma] * \{\sigma : \tau\} \leq \tau * \{\sigma : \tau\} \qquad (45)$$

Figure 7. Some subtyping axioms

and write $\iota : \mathit{Cap} \to W$ for the isomorphism in (46), and ι^{-1} for its inverse. By ordering the elements of *Cap* pointwise,

$$p \leq q \;\Leftrightarrow\; \forall w .\, p(w) \subseteq q(w),$$

we can lift the algebra structure on $\mathit{UPred}^{\uparrow}(\mathit{Heap})$.

Lemma 3.1. With the above ordering and the pointwise lifting of the algebra operations on $\mathit{UPred}^{\uparrow}(\mathit{Heap})$, the set *Cap* is a complete Heyting BI algebra.

The fact that *Cap* is a complete BI algebra immediately gives us a sound interpretation of $*$ on capabilities. (Moreover, it suggests that the syntax of capabilities could be extended with all the logical connectives of separation logic.) However, to interpret recursive capabilities we also need to know that the operations are non-expansive:

Lemma 3.2. The BI algebra operations on *Cap* are non-expansive functions, i.e., they are morphisms in $\mathbf{CBUlt}_{\mathrm{ne}}$:

$$\wedge, \vee, \to, *, -\!\!* : \mathit{Cap} \times \mathit{Cap} \to \mathit{Cap}$$

$$\textstyle\bigwedge_I, \bigvee_I : (I \to \mathit{Cap}) \to \mathit{Cap}$$

(For the last two operations, the indexing set I is given the discrete metric, i.e., the distance of any two different elements is 1.)

Proof sketch. One can first show the corresponding property for the operations on $\mathit{UPred}^{\uparrow}(\mathit{Heap})$ which is straightforward; the result then follows from the pointwise definition and the use of the sup-metric on *Cap*. To illustrate the non-expansiveness on $\mathit{UPred}^{\uparrow}(\mathit{Heap})$, we consider the case of separating conjunction: It suffices to show that $p \overset{n}{=} p'$ and $q \overset{n}{=} q'$ implies $p * q \overset{n}{=} p' * q'$. By definition of the n-equality, $p * q \overset{n}{=} p' * q'$ is equivalent to $(j, h) \in p * q \;\Leftrightarrow\; (j, h) \in p' * q'$ for all $j < n$, which follows easily from the assumptions that $p \overset{n}{=} p'$ and $q \overset{n}{=} q'$. □

Next, we define a 'composition' operation on the worlds W. This operation plays a role similar to the ordering by extension in the case where worlds are finite maps from locations to semantic types (cf. Section 2). However, it is more involved than a simple extension of worlds; rather, it corresponds to the syntactic abbreviation $C_1 \circ C_2 = C_1 \otimes C_2 * C_2$ from Figure 6, of conjoining C_1 and C_2 and additionally applying an invariant extension $\cdot \otimes C_2$ to C_1. Formally, $\circ : W \times W \to W$ is a non-expansive operation that for all $p, r, w \in W$ satisfies

$$\iota^{-1}(p \circ r)(w) \;=\; \iota^{-1}(p)(r \circ w) \;*\; \iota^{-1}(r)(w)\,.$$

Using the metric-space setup, we can define this operation by an easy application of Banach's fixed point theorem, as in [44]. Observe that it is here where we exploit that we have obtained a proper solution to the world equation (46) in $\mathbf{CBUlt}_{\mathrm{ne}}$.

We write *emp* for the image $\iota(\lambda w. I)$ of the BI unit under ι. Then it turns out that $\circ$ is associative and $p \circ \mathit{emp} = \mathit{emp} \circ p = p$ holds for all p, so $(W, \circ, \mathit{emp})$ is a monoid in $\mathbf{CBUlt}_{\mathrm{ne}}$. Now let (X, d) be an arbitrary ultrametric space. Using the composition operator on worlds, we consider a semantic analogue of the invariant extension operation, $\otimes : X^{(\frac{1}{2}\cdot W)} \times W \to X^{(\frac{1}{2}\cdot W)}$ defined by

$$(f \otimes w_0)(w) \;=\; f(w_0 \circ w)\;.$$

The following proposition is a slight generalization of [44, Lemma 5], and summarizes the key properties of $\circ$ and $\otimes$. In the following, these properties are used to justify some of the equivalences given in Figure 6.

Proposition 3.3 (Monoid and monoid action)**.** Let (X, d) be an ultrametric space. Then $(W, \circ, \mathit{emp})$ is a monoid in $\mathbf{CBUlt}_{\mathrm{ne}}$, and the operation $\otimes : X^{(\frac{1}{2}\cdot W)} \times W \to X^{(\frac{1}{2}\cdot W)}$ is a (non-expansive) action of the monoid W on the ultrametric space of non-expansive functions from $\frac{1}{2}\cdot W$ to X, i.e., $f \otimes \mathit{emp} = f$ and $(f \otimes w_1) \otimes w_2 = f \otimes (w_1 \circ w_2)$.

3.3 Semantic Domains and Interpretation

In this section we give the interpretation of the capabilities, types and environments of the type system. The semantic domain corresponding to each syntactic category is a set of (contractively world-dependent) upwards closed and uniform predicates:

$$\begin{aligned} \mathit{VT} &= \tfrac{1}{2}\cdot W \to \mathit{UPred}^{\uparrow}(\mathit{Val}) \\ \mathit{MT} &= \tfrac{1}{2}\cdot W \to \mathit{UPred}^{\uparrow}(\mathit{Val} \times \mathit{Heap})\;. \end{aligned}$$

In particular, in each case there is an action of W by the operation $\otimes$, as described in Proposition 3.3. Note that $\mathit{Cap} = \frac{1}{2}\cdot W \to \mathit{UPred}^{\uparrow}(\mathit{Heap})$ acts on itself, via the isomorphism ι between W and *Cap*. This operation plays a key role in explaining the higher-order frame (and also anti-frame) inference rules and the associated distribution axioms [44, 46]. Moreover, due to the shrinking factor $\delta = \frac{1}{2}$, this action is contractive in its right-hand side: for all $p, r \in \mathit{Cap}$, the assignment $r \mapsto p \otimes \iota(r)$ is a contractive endomap on *Cap*. This observation explains why the (syntactic) invariant extension can be assumed formally contractive in its second argument.

We also consider a further overloading of the separating conjunction. It is the below generalization $S * q$ to sets of the form $S \in \mathit{UPred}^{\uparrow}(A \times \mathit{Heap})$ and $q \in \mathit{UPred}^{\uparrow}(\mathit{Heap})$:

$$S * q = \{(k, (a, h \cdot h')) \mid (k, (a, h)) \in S \wedge (k, h') \in q \wedge h \# h'\}.$$

As for the separating conjunction on $\mathit{UPred}^{\uparrow}(\mathit{Heap})$, this operation can be lifted pointwise to give a non-expansive operation on $S \in \frac{1}{2}\cdot W \to \mathit{UPred}^{\uparrow}(A \times \mathit{Heap})$ and $r \in \mathit{Cap}$,

$$(S * r)(w) = S(w) * r(w)\;. \tag{47}$$

This provides a second monoid action, with respect to the monoid structure given by the separating conjunction on *Cap*.

Proposition 3.4 (Monoid and monoid action)**.** $(\mathit{Cap}, *, I)$ is a commutative monoid, and for any (pre-ordered) set A the operation in (47) is an action of this monoid on the space of non-expansive functions from $\frac{1}{2}\cdot W$ to $\mathit{UPred}^{\uparrow}(A \times \mathit{Heap})$, i.e., $S * I = S$ and $(S * p) * q = S * (p * q)$.

The interpretation of capabilities and types is given in Figure 9. This 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 $\eta(\beta) \in \mathit{MT}$. As indicated above, the semantics of capabilities is defined in terms of the BI structure on *Cap*. The semantics of memory types uses the action of *Cap* on *MT* described in (47). It also makes explicit the aliasing information contained in memory types: for instance, the two components of a pair of type $\theta_1 \times \theta_2$ cannot overlap in the heap (a similar exclusion of sharing holds for referenced cells). In the interpretation of a value type τ considered as memory type, $[\![\tau]\!]$ on the right-hand side refers to the value type interpretation. Note that the computation types χ form a subset of the memory types, and thus obtain their interpretation in *MT*.

Let *Env* denote the finite maps from variables to closed values. Duplicable (heap-independent) environments are interpreted as contractive maps from W to $\mathit{UPred}^{\uparrow}(\mathit{Env})$. Linear environments are modelled as contractive maps from W to $\mathit{UPred}^{\uparrow}(\mathit{Env} \times \mathit{Heap})$. Conceptually, each of the entries in a linear environment owns a part of the heap, disjoint from that of the other entries.

With the exception of arrow types, the semantics of value types deserves little explanation; in all cases, the world is simply passed on, and the index is decreased (whenever justified by the operational semantics) to ensure that type constructors become contractive. The definition of arrow types is more intricate, and uses the following extension of memory types from values to expressions.

Definition 3.5. Let $S \in \mathit{MT}$. Then the function $\mathcal{E}(S) : W \to \mathit{UPred}^{\uparrow}(\mathit{Exp} \times \mathit{Heap})$ is defined by $(k, (t, h)) \in \mathcal{E}(S)(w)$ iff

$$\begin{aligned} &\forall j \le k, t', h'.\ (t \mid h) \longmapsto^{j} (t' \mid h') \;\wedge\; (t' \mid h') \text{ irreducible} \\ &\qquad\qquad \Rightarrow\; (k - j, (t', h')) \in S(w) * \iota^{-1}(w)(\mathit{emp})\;. \end{aligned}$$

Note that there is no scaling by $\frac{1}{2}$, i.e., $\mathcal{E}(S)$ is a non-expansive, but not a contractive, function of worlds. However, we do have a form of contractiveness on non-values:

Lemma 3.6. For all $S_1, S_2 \in \mathit{MT}$, expressions t and $h \in \mathit{Heap}$, if $w_1 \overset{n}{=} w_2$ in W, $S_1 \overset{n}{=} S_2$ and $t \notin \mathit{Val}$, then for all $k \le n$,

$$(k, (t, h)) \in \mathcal{E}(S_1)(w_1) \;\Leftrightarrow\; (k, (t, h)) \in \mathcal{E}(S_2)(w_2)\;.$$

Proof. Let $w_1 \overset{n}{=} w_2$, and observe that this implies $S_1(w_1) \overset{n}{=} S_2(w_2)$ and $\iota^{-1}(w_1)(\mathit{emp}) \overset{n}{=} \iota^{-1}(w_2)(\mathit{emp})$. Now assume that $(k, (t, h)) \in \mathcal{E}(S_1)(w_1)$ for some $k \le n$. We must show that $(k, (t, h)) \in \mathcal{E}(S_2)(w_2)$. For this, suppose that $(t \mid h) \longmapsto^{j} (t' \mid h')$ for some $j \le k$ where $(t' \mid h')$ is irreducible. The assumption $(k, (t, h)) \in \mathcal{E}(S_1)(w_1)$ yields $(k - j, (t', h')) \in S_1(w_1) * \iota^{-1}(w_1)(\mathit{emp})$. In particular, $t' \in \mathit{Val}$ and therefore $t \neq t'$ by the assumption that $t \notin \mathit{Val}$. Thus we must have $j > 0$, and therefore $k - j < k \le n$ which by the above observations means that $(k - j, (t', h')) \in S_2(w_2) * \iota^{-1}(w_2)(\mathit{emp})$.

The direction from right to left is symmetric. □

We now explain the ideas behind the definition of arrow types in Figure 9 in more detail. First, the basic idea of our Kripke style semantics is that invariants added by the context are collected in the worlds. Thus, for a procedure application we realize this idea by interpreting the current world as a predicate $\iota^{-1}(w)(\mathit{emp})$ on heaps, which is conjoined to the actual argument (computation) type $[\![\chi_1]\!]_\eta\,(w)$, as well as to the result (computation) type $[\![\chi_2]\!]_\eta\,(w)$ through the definition of $\mathcal{E}$. Second, by additionally conjoining r as an invariant we bake in the first-order frame property. Finally, the quantification over indices j less than k achieves that $[\![\chi_1 \to \chi_2]\!]_\eta\, w$ is in $\mathit{UPred}^{\uparrow}(\mathit{Val})$. There are two explanations why we require that j be *strictly* less than k in the definition of $[\![\chi_1 \to \chi_2]\!]$. Technically, since $\iota^{-1}(w)(\mathit{emp})$ is non-expansive but not contractive in w, we need the strict inequality to ensure the non-expansiveness of $[\![\chi_1 \to \chi_2]\!]$ as a function $\frac{1}{2}\cdot W \to \mathit{UPred}^{\uparrow}(\mathit{Val})$. Moreover, the smaller index allows us to prove the typing rule for recursive functions, by induction on k. Intuitively, the use of $j < k$ for the arguments suffices since each procedure application consumes a step. The final $j + 1$ index on expressions in the interpretation of arrow types is justified by the above Lemma.

Proposition 3.7. The interpretation in Figure 9 is well-defined: all the $[\![\cdot]\!]$'s map into the declared sets, and the recursive definitions of capabilities and types have unique solutions.

Capabilities, $\llbracket C \rrbracket_\eta : \frac{1}{2} \cdot W \to \mathit{UPred}^{\uparrow}(\mathit{Heap})$

$$\begin{aligned}
\llbracket C_1 \otimes C_2 \rrbracket_\eta\, w &= (\llbracket C_1 \rrbracket_\eta \otimes \iota(\llbracket C_2 \rrbracket_\eta))w \\
\llbracket \emptyset \rrbracket_\eta\, w &= \mathbb{N} \times \mathit{Heap} \\
\llbracket C_1 * C_2 \rrbracket_\eta\, w &= (\llbracket C_1 \rrbracket_\eta * \llbracket C_2 \rrbracket_\eta)w \\
\llbracket \{\sigma : \theta\} \rrbracket_\eta\, w &= \{(k,h) \mid (k,(\eta(\sigma),h)) \in \llbracket \theta \rrbracket_\eta\, w\} \\
\llbracket \exists \sigma.C \rrbracket_\eta\, w &= \textstyle\bigcup_{v \in \mathit{Val}} \llbracket C \rrbracket_{\eta[\sigma:=v]}\, w \\
\llbracket \gamma \rrbracket_\eta\, w &= \eta(\gamma)(w) \\
\llbracket \mu\gamma.C \rrbracket_\eta\, w &= \mathit{fix}(\lambda r.\ \llbracket C \rrbracket_{\eta[\gamma:=r]})w
\end{aligned}$$

Value types, $\llbracket \tau \rrbracket_\eta : \frac{1}{2} \cdot W \to \mathit{UPred}^{\uparrow}(\mathit{Val})$

$$\begin{aligned}
\llbracket \tau \otimes C \rrbracket_\eta\, w &= (\llbracket \tau \rrbracket_\eta \otimes \iota(\llbracket C \rrbracket_\eta))w \\
\llbracket 0 \rrbracket_\eta\, w &= \emptyset \\
\llbracket 1 \rrbracket_\eta\, w &= \mathbb{N} \times \{()\} \\
\llbracket \mathsf{int} \rrbracket_\eta\, w &= \mathbb{N} \times \{\underline{n} \mid n \in \mathbb{Z}\} \\
\llbracket \tau_1 + \tau_2 \rrbracket_\eta\, w &= \{(k, \mathsf{inj}^i v) \mid k > 0 \ \Rightarrow\ (k{-}1, v) \in \llbracket \tau_i \rrbracket_\eta\, w\} \\
\llbracket \tau_1 \times \tau_2 \rrbracket_\eta\, w &= \{(k, (v_1, v_2)) \mid k > 0 \ \Rightarrow\ (k{-}1, v_i) \in \llbracket \tau_i \rrbracket_\eta\, w\} \\
\llbracket \chi_1 \to \chi_2 \rrbracket_\eta\, w &= \{(k, v) \mid \forall j < k.\ \forall r \in \mathit{Cap}. \\
&\qquad \forall (j, (v', h)) \in (\llbracket \chi_1 \rrbracket_\eta * r)(w) * \iota^{-1}(w)(\mathit{emp}). \\
&\qquad\quad (j{+}1, (v\, v', h)) \in \mathcal{E}(\llbracket \chi_2 \rrbracket_\eta * r)(w)\} \\
\llbracket [\sigma] \rrbracket_\eta\, w &= \mathbb{N} \times \{\eta(\sigma)\} \\
\llbracket \alpha \rrbracket_\eta\, w &= \eta(\alpha)(w) \\
\llbracket \mu\alpha.\tau \rrbracket_\eta\, w &= \mathit{fix}(\lambda S.\ \llbracket \tau \rrbracket_{\eta[\alpha:=S]})w \\
\llbracket \forall\alpha.\tau \rrbracket_\eta\, w &= \textstyle\bigcap_{a \in \mathit{VT}} \llbracket \tau \rrbracket_{\eta[\alpha:=a]}\, w
\end{aligned}$$

Memory types, $\llbracket \theta \rrbracket_\eta : \frac{1}{2} \cdot W \to \mathit{UPred}^{\uparrow}(\mathit{Val} \times \mathit{Heap})$

$$\begin{aligned}
\llbracket \theta \otimes C \rrbracket_\eta\, w &= (\llbracket \theta \rrbracket_\eta \otimes \iota(\llbracket C \rrbracket_\eta))\, w \\
\llbracket \tau \rrbracket_\eta\, w &= \{(k, (v, h)) \mid h \in \mathit{Heap},\ (k, v) \in \llbracket \tau \rrbracket_\eta\, w\} \\
\llbracket \theta_1 + \theta_2 \rrbracket_\eta\, w &= \{(k, (\mathsf{inj}^i v, h)) \mid k{>}0 \ \Rightarrow\ (k{-}1, (v, h)) \in \llbracket \theta_i \rrbracket_\eta\, w\} \\
\llbracket \theta_1 \times \theta_2 \rrbracket_\eta\, w &= \{(k, (v_1, v_2), h_1{\cdot}h_2) \mid k{>}0 \Rightarrow (k{-}1, (v_i, h_i)) {\in} \llbracket \theta_i \rrbracket_\eta w\} \\
\llbracket \mathsf{ref}\,\theta \rrbracket_\eta\, w &= \{(k, (l, h{\cdot}[l \mapsto v])) \mid k{>}0 \Rightarrow (k{-}1, (v, h)) {\in} \llbracket \theta \rrbracket_\eta w\} \\
\llbracket \theta * C \rrbracket_\eta\, w &= (\llbracket \theta \rrbracket_\eta\, w) * (\llbracket C \rrbracket_\eta\, w) \\
\llbracket \exists\sigma.\theta \rrbracket_\eta\, w &= \textstyle\bigcup_{v \in \mathit{Val}} \llbracket \theta \rrbracket_{\eta[\sigma:=v]}\, w \\
\llbracket \beta \rrbracket_\eta\, w &= \eta(\beta)(w) \\
\llbracket \mu\beta.\theta \rrbracket_\eta\, w &= \mathit{fix}(\lambda S.\ \llbracket \theta \rrbracket_{\eta[\beta:=S]})w
\end{aligned}$$

Duplicable environments, $\llbracket \Delta \rrbracket_\eta : \frac{1}{2} \cdot W \to \mathit{UPred}^{\uparrow}(\mathit{Env})$

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

Linear environments, $\llbracket \Gamma \rrbracket_\eta : \frac{1}{2} \cdot W \to \mathit{UPred}^{\uparrow}(\mathit{Env} \times \mathit{Heap})$

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

Figure 9. Interpretation

Proof sketch. We equip the set of values with the discrete metric, and then obtain a complete 1-bounded ultrametric on environments:

$$d(\eta, \eta') \ = \ \sup_\xi d(\eta(\xi), \eta'(\xi)) \,. \tag{48}$$

We then show by simultaneous induction on C, τ, and θ, the following properties:

1. $\llbracket C \rrbracket_\eta\, w$, $\llbracket \tau \rrbracket_\eta\, w$, and $\llbracket \theta \rrbracket_\eta\, w$ are upwards closed and uniform predicates;
2. $\llbracket C \rrbracket_\eta\, w$, $\llbracket \tau \rrbracket_\eta\, w$, and $\llbracket \theta \rrbracket_\eta\, w$ are non-expansive functions of η (with respect to the distance in (48)) and w (with respect to the metric on $\frac{1}{2} \cdot W$);
3. if C is formally contractive in ξ then $\llbracket C \rrbracket_{\eta[\xi:=(\cdot)]}$ is contractive;
4. if θ is formally contractive in ξ then $\llbracket \theta \rrbracket_{\eta[\xi:=(\cdot)]}$ is contractive.

These properties can be verified by a straightforward (but tedious) simultaneous induction, for instance using Lemma 3.6 and the non-expansiveness of separating conjunction to show the non-expansiveness of arrow types. The interpretation of recursive types and capabilities relies on our restriction to formally contractive equations, so that they are uniquely defined from Banach's fixed point theorem by the above properties 3 and 4. □

This interpretation respects the structural equivalence, i.e., whenever C_1 and C_2 are equivalent capabilities then $\llbracket C_1 \rrbracket = \llbracket C_2 \rrbracket$ (and similarly for value and memory types). The proofs of these facts are easy consequences of the definition of $\llbracket C \rrbracket$ and Propositions 3.3 and 3.4. Moreover, the interpretation validates the subtyping axioms, i.e., whenever $\theta_1 \leq \theta_2$ then $\llbracket \theta_1 \rrbracket_\eta\, w \subseteq \llbracket \theta_2 \rrbracket_\eta\, w$ holds for all η and w. These proofs can be found in the appendix of the long version of the paper.

Recall that we have two kinds of judgments, one for typing of values and the other for the typing of expressions:

$$\Delta \vdash v : \tau \qquad\qquad \Gamma \Vdash t : \chi$$

The semantics of a value judgement simply establishes truth with respect to all worlds w, all environments η and all $k \in \mathbb{N}$:

$$\begin{aligned}
&\models (\Delta \vdash v : \tau) \ \overset{\mathit{def}}{\iff} \\
&\quad \forall\eta.\ \forall w \in W.\ \forall k \in \mathbb{N}.\ \forall (k, \rho) \in \llbracket \Delta \rrbracket_\eta\, w.\ (k, \rho(v)) \in \llbracket \tau \rrbracket_\eta\, w\,.
\end{aligned}$$

Here $\rho(v)$ means the application of the substitution ρ to v. The judgement for expressions mirrors the interpretation of the arrow case for value types, in that there is also a quantification over heap predicates $r \in \mathit{Cap}$:

$$\begin{aligned}
&\models (\Gamma \Vdash t : \chi) \ \overset{\mathit{def}}{\iff} \\
&\quad \forall\eta.\ \forall w \in W.\ \forall k \in \mathbb{N}.\ \forall r \in \mathit{Cap}. \\
&\qquad \forall (k, (\rho, h)) \in (\llbracket \Gamma \rrbracket_\eta * r)w * \iota^{-1}(w)(\mathit{emp}). \\
&\qquad\quad (k, (\rho(t), h)) \in \mathcal{E}(\llbracket \chi \rrbracket_\eta * r)(w).
\end{aligned}$$

The universal quantifications allow us to have frame rules: the universal quantification over worlds w ensures the soundness of the deep frame rule, and the universal quantification over capabilities r validates the shallow frame rule.

We can now give the main result of this section, which expresses that the extension of the capability system with higher-order frame rules is sound. In particular, the below theorem implies type safety.

Theorem 3.8 (Soundness)**.** If $\Delta \vdash v : \tau$ then $\models (\Delta \vdash v : \tau)$, and if $\Gamma \Vdash t : \chi$ then $\models (\Gamma \Vdash t : \chi)$.

To prove the theorem, we show that each typing rule preserves the truth of judgements. These proofs are given in the appendix of the long version of the paper.

3.4 Observations

We conclude this section with some remarks on the model.

Structural equivalence. In previous work, where Pottier first introduced the anti-frame rule [39], the syntactic types are considered modulo the structural equivalence. This means that they are not inductively defined, and consequently Pottier avoids inductive proofs on their syntax. In contrast, our interpretation is given by induction on the structure of types and capabilities, and only after having established the interpretation do we consider the structural equivalence (and prove that our interpretation respects it).

Unique solutions proof principle. In practice, one may have to show type equivalences that do not easily follow from the structural equivalence. The metric structure of our model suggests a proof principle for this, by the uniqueness of solutions of contractive type equations: if two types are solutions of a common contractive fixed point equation, then we can conclude that they are equal.

Additional subtyping axioms. Our model satisfies some additional subtyping axioms that have not been mentioned in the literature before. These refer, e.g. to the duplication of value capabilities. In particular, our model implies the soundness of the axiom

$$[\sigma] * \{\sigma : \tau\} \;\leq\; [\sigma] * \{\sigma : \tau\} * \{\sigma : \tau\}.$$

A possible explanation why these axioms have not been noted before may be that previous soundness proofs for capability type systems (e.g. by translation [24] or progress and preservation [39]) rest on invariants that are stronger than necessary.

Classical interpretation. The capability calculus of Crary et al. [26] has a memory deallocation construct, and satisfies a "complete collection" property. Essentially, if a program of type $\tau * \emptyset$ terminates, then it does so in an empty heap. After dropping the subtyping axiom $C_1 * C_2 \leq C_1$ and adding a deallocation construct to our calculus, it would also satisfy this property. Our approach is flexible enough so that this can be shown by modifying the semantics and using the "classical" interpretation of separation logic. That is, the definition of worlds and capabilities would be based on $\mathit{UPred}(\mathit{Heap})$ instead of $\mathit{UPred}^{\uparrow}(\mathit{Heap})$, where the BI structure is given by $*$ and $-\!\!*$ as above but with unit $I = \omega \times \{[]\}$. However, this complete collection property is not only destroyed by the axiom $C_1 * C_2 \leq C_1$ but also by the inclusion of an anti-frame rule (which we do not consider in this paper, though).

Aliasing. Even though our model is based on the operational semantics, it gives a semantic understanding of capabilities. Let θ be any memory type, and consider the recursively defined memory type $\mathit{mlist} = \mu\beta.\mathsf{ref}\ 1 + \theta \times \beta$ of mutable lists from [24]. In *loc. cit.* it is mentioned (without proof) that this is the type of mutable *non-aliased* lists, and our semantics shows very directly that this is indeed the case: From the semantics of memory types in Figure 9, we see (for k large enough) that $(k, (l_1, h)) \in [\![\mathit{mlist}]\!]$ just in case that for some $n < k$, heap h can be split up into $2n$ disjoint parts:

$$h = [l_1 \mapsto (v_1, l_2)] \cdot h_1 \cdot [l_2 \mapsto (v_2, l_3)] \cdot h_2 \cdots [l_n \mapsto ()] \cdot h_n,$$

with $(k{-}i, (v_i, h_i)) \in [\![\theta]\!]$, for all $1 \leq i < n$. Thus, all list entries live in disjoint parts of the heap and all locations l_i must be distinct; in particular, mlist cannot contain cyclic lists.

A domain-theoretic model? Finally, it may be worthwhile to remark that we have tried to give a domain-theoretic ultrametric model of the calculus as well, but found this very difficult. In fact, this difficulty was one motivation for constructing the present step-indexed model. Very roughly, the projections that come with solutions to recursive domain equations disagree with, e.g., memory types. The step-indexed setting appears a bit more flexible in this respect.

4. Specialization to Indirection Theory

Hobor, Dockins and Appel [30] present a general *theory of indirection* for giving set-theoretic models of recursively defined structures. Faced with a recursive equation, Hobor et al. provide an approximate solution: this is a set together with a pair of functions characterized by the two axioms of indirection theory that elegantly capture the approximative nature of the solution.

Our approach to recursive equations is different. We provide an exact solution, but in a category of metric spaces instead of the category of sets and functions. In this section we argue that our approach is more general in the sense that, for the same recursive equation, one may build the approximative solution of Hobor et al. from our solution.

Before starting, we point out that this specialization to indirection theory is not unconditional. The construction presented by Hobor et al. is parameterized over a set-theoretic functor F : **Set** $\to$ **Set**, and this functor must in a suitable sense have an extension to $\mathbf{CBUlt}_{\mathrm{ne}}$ in order for our approach to apply. Fortunately, this condition holds for functors on **Set** built with standard constructors.

We now sketch how the specialization to indirection theory proceeds. The full story, including proofs, can be found in the appendix of the long version of the paper.

4.1 Indirection Theory

Assume that we are given a functor F : **Set** $\to$ **Set** and a non-empty set O. Let $2 = \{0, 1\}$ be the set of "truth values." Indirection theory begins from the desire to solve the equation

$$K \cong F(K \times O \to 2) \tag{49}$$

in **Set**, which is often impossible for cardinality reasons.[6] Instead, one obtains an approximate solution

$$\begin{tikzcd} K \arrow[r, shift left, "\mathsf{unsquash}"] & \mathbb{N} \times F(K \times O \to 2) \arrow[l, shift left, "\mathsf{squash}"] \end{tikzcd}$$

consisting of a set K and functions squash and $\mathsf{unsquash}$ satisfying:

1. $\mathsf{squash}(\mathsf{unsquash}\ k) = k$.
2. $\mathsf{unsquash}(\mathsf{squash}(m, \nu)) = (m, F(\mathsf{approx}_m)(\nu))$.

Here $\mathsf{level} = \mathsf{fst} \circ \mathsf{unsquash} : K \to \mathbb{N}$, and the map $\mathsf{approx}_m : (K \times O \to 2) \to (K \times O \to 2)$ is defined, for each $m \in \mathbb{N}$, by

$$\mathsf{approx}_m(\psi)(k, o) \;=\; (\psi(k, o) \wedge \mathsf{level}(k) < m).$$

The idea is that elements of K have "levels," and that the function approx_m transforms a predicate on K to one that only holds for elements of level less than m. Notice that squash is a left inverse of $\mathsf{unsquash}$, but in general not a right inverse: $\mathsf{unsquash}(\mathsf{squash}(m, \nu))$ is in some sense an approximation of (m, ν).

4.2 From Metric Spaces to Indirection Theory

Every set can be considered as a metric space by giving it the discrete metric d (i.e., $d(x, y) = 1$ if $x \neq y$). In this way, the category of non-empty sets can be viewed as a subcategory of $\mathbf{CBUlt}_{\mathrm{ne}}$. We now assume that the functor F : **Set** $\to$ **Set** considered above has a so-called *plain lift* $\hat{F} : \mathbf{CBUlt}_{\mathrm{ne}} \to \mathbf{CBUlt}_{\mathrm{ne}}$. This means that $\hat{F}$ is a locally non-expansive functor which agrees with F on non-empty sets (and functions between them), and also that $\hat{F}$ satisfies some technical conditions given in the appendix of the long version of

[6] Unlike Hobor et al. we do not parameterize over the set of truth values. The generalization, while probably technically feasible, does not appear necessary for applications.

the paper. As noted above, plain lifts exist for all the standard constructors (see again the long version of the paper). In particular we have plain lifts of the functors of all the examples of Hobor et al.[7]

From Theorem 2.1 and Lemma 2.4, we easily obtain:

Theorem 4.1. Let F : **Set** → **Set** be a functor with a plain lift $\hat{F}$: **CBUlt**$_{\text{ne}}$ → **CBUlt**$_{\text{ne}}$. There is a non-empty, complete, 1-bounded ultrametric space X and an isomorphism

$$\Phi \;:\; X \;\;\cong\;\; \hat{F}\left(\tfrac{1}{2}\left(X \to \mathit{UPred}(O)\right)\right),$$

where the function space consists of non-expansive maps.

We now show that one can use such an isomorphism to construct an approximate solution in the sense of indirection theory.

We deviate from Hobor et al. by building a solution that features only so-called *hereditary* maps from $K \times O$ to 2. This is a direct consequence of the downwards closedness required of members of $\mathit{UPred}(O)$, since hereditary predicates are, intuitively, “closed under approximation” in the K component. We regard this difference as an improvement since, in all applications to date, hereditariness is considered desirable and often necessary. This point is discussed further below.

Theorem 4.2. Let F : **Set** → **Set** be a functor with a plain lift $\hat{F}$: **CBUlt**$_{\text{ne}}$ → **CBUlt**$_{\text{ne}}$. We can, from the isomorphism of Theorem 4.1, build a set K, a subset of *hereditary* maps $K \times O \to_{her} 2$ of the full function space $K \times O \to 2$ and two maps

$$\begin{tikzcd}
K \arrow[r, shift left, "\text{unsquash}"] & \mathbb{N} \times F(K \times O \to_{her} 2) \arrow[l, shift left, "\text{squash}"]
\end{tikzcd}$$

satisfying Hobor et al.’s requirements for an approximate solution (items 1 and 2 above).

Advantages of metric solution approach. Having proved that our metric-space approach specializes to the indirection theory, we now proceed to argue some advantages of our approach in general.

Firstly, although we do not think that the step-indexed version of our metric-space approach is more expressive than standard step-indexed models, we believe that our version provides a good framework for doing step-indexing with useful conceptual guidelines. This holds even if we disregard recursively defined worlds. Consider, e.g., the interpretation of recursive types in Figure 9 of Section 3. Under the aforementioned assumptions of formal contractiveness, we are free to interpret recursive types as actual fixed points – a conceptually pleasing thing to do. Another example is the ⊗ operator in the same section, which is also constructed using Banach’s fixed-point theorem. A similar construction could possibly be pushed through either with hand-built approximate worlds as employed by Ahmed et al. [6], with the indirection theory of Hobor et al. [30] or the higher order μ described in Hobor’s thesis [28]. But the precise course of action is much less immediate.

Secondly, in comparison with the indirection theory [30], our approach of solving recursive metric equations allows one to use a body of supporting theory on metric spaces and to construct a wider variety of possible worlds to be used in Kripke models. To illustrate this point, let us focus on the step-indexed model of ML references discussed in Section 2.3 and in Sections 2.1, 4.1 and 5 of [30]. In the model provided by indirection theory, types are arbitrary maps from worlds to values, modulo currying and nomenclature. But, as argued in [30, Section 5.1], we really want types that are both hereditary and monotone. In [30, Section 5.1], such types are elegantly identified using modal operators, but this does not change the problem that the types in a world may fail to meet these criteria. This is addressed in the last paragraph of [30, Section 10] where an alternative, and less straightforward, model with only hereditary types in the worlds is sketched. Alas, this means that one has to start the model construction all over again from scratch and it does not buy us monotonicity[8]. On the other hand, to obtain hereditary types with the metric approach we just use the downwards-closure condition on $\mathit{UPred}(V)$, verify Lemma 2.4 and apply Theorem 2.1. And to work with monotone types, we can apply a slightly stronger existence result [20, Proposition 5.4] for *pre-ordered* metric spaces. By a similar argument one can extend the approach to mixed variance functors discussed in [30, Section 10]: Indeed, in unpublished work we have used mixed-variance functors to verify that the metric-space approach scales to the elaborate worlds of [6].

Finally, we think that it is advantageous that the metric approach applies both to models based on domain theory and to models based on operational semantics.

5. Related and Future Work

Relational reasoning. We have focused on unary reasoning in this paper, but the techniques developed here also apply to relational reasoning. Relational reasoning principles about programs with higher-order store, such as logical relations for reasoning about contextual equivalence of programs, have been developed both based on domain theory (e.g., [15, 21]), and on step-indexed models (e.g., [6]). For such relational reasoning, the worlds are typically more sophisticated than the worlds we have discussed so far. This is because for relational reasoning worlds need to describe situations in which programs are contextually equivalent even though they use local states in different ways. One of us (Thamsborg) has recently phrased the state-of-the-art world model from [6] as a recursive world equation over a domain-theoretic model. He did this to obtain more abstract proof principles for program equivalences, which does not involve reasoning about step indices. Alternatively, Dreyer et al. [27] have shown how to extend the relational step-indexed model [6] to a model of a modal logic for more abstract reasoning about program equivalences. The latter modal logic has been derived from the step-indexed model. Even with this development, it is still a challenge to develop relational step-indexed models of Hoare Type Theory [35] and its new developments. It would be interesting to see whether the step-indexed metric space approach can be used to address this challenge.

Formalization. An often mentioned advantage of the traditional step-indexed approach is that it lends itself well to formalization in theorem provers. Indeed, impressive formalization work has been carried out in, e.g., Coq [10].

Thus, one may wonder whether our proposed metric approach hinders formalizations. It does not. Following the treatment in [20], Varming et al. have given a constructive formalization of the solutions of recursive metric-space equations in Coq [16], including the step-indexed model of ML references from Section 2.3.

Capabilities. In [5], Ahmed et al. presented a step-indexed model of a substructural type system, which is similar to the capability calculus considered in this paper. However, their model did not provide a satisfactory semantic analysis of capabilities. Ahmed et al. instrumented the operational semantics with abstract run-time entities corresponding to capabilities, and their model included those abstract entities, instead of giving a semantic analysis of what they really should denote. Moreover, they did not consider non-trivial combinations of capabilities such as $C_1 * C_2$ and did not

[7] With the possible exception of Example 2.7. The functor in that example is complex, and the presentation is a bit dense, so we are not sure whether the functor has a plain lift.

[8] It is worth noticing that since the publication of [30], Hobor et al. have given an even more expressive notion of indirection theory that guarantee monotone types too and works with mixed variance functors, see [10].

include frame rules, etc. The step-indexed model in this paper does not alter the operational semantics, interprets capabilities including $C_1 * C_2$, and justifies (shallow and deep) frame rules.

We point out that an alternative semantic model of the basic capability system could be obtained by combining the functional translation of Charguéraud and Pottier [24] with a semantic model of their purely functional target calculus. The functional translation in [24] does not, however, include higher-order frame rules and it is not immediate how to include those rules.

To extend our semantics to group regions is future work. Note that group regions are non-trivial, since they might grow over time but types need to be invariant (monotone) with respect to this growth. Further extensions will address, for instance, frame rules for more general (parameterized) invariants on local state [38].

Other operational techniques. We briefly mention two techniques other than step indexing that can be used to define logical relations based on operational semantics. First, *syntactic minimal invariance* [18, 25] is based on operational counterparts of the projection functions one obtains from solutions to recursive domain equations. As far as we know, this technique has not been developed for languages with store. Second, *biorthogonality* [13, 31, 37] is based on syntactically defined closure operators on relations. Biorthogonality has been developed for a language with integer store [37], but not (without also using step indexing) for languages with general recursive types or higher-order store. Voullion and Melliès [50] give an axiomatic setup that incorporates both techniques for a language without store. See also [34] by the same authors.

As an alternative to logical relations, techniques based on *bisimulation* can be used to show contextual equivalences for languages with store [49]. However, such techniques do not seem helpful for modelling expressive type systems such as the one considered in Section 3.

6. Conclusion

In this paper, we have argued that recursive features of programming languages, type systems and program logics, such as higher-order store, can be naturally interpreted via Kripke models over worlds that are recursively defined in a category of metric spaces. Interestingly, this can be carried out not only denotationally but also using operational semantics. Our method combines the simplicity of existing step-indexed models with the accuracy of domain-theoretic approaches for recursive domain equations. Unlike other step-indexed models, our method uses solutions of the original recursive equations, not their approximated versions. The benefits of this technique have been demonstrated in our new semantics of Charguéraud and Pottier's type-and-capability system [24], where solving an original recursive equation over worlds played a seemingly crucial role in modelling a recursively defined operator on worlds.

Additionally, we have shown that our metric approach can be specialized to Hobor et al.'s recent proposal [30] and argued that the metric approach has some advantages.

Acknowledgments

We would like to thank François Pottier, Aquinas Hobor, Robert Dockins, Andrew W. Appel and Carsten Varming for helpful discussions and insightful comments. Yang and Reus acknowledge support from the EPSRC.

References

[1] M. Abadi and G. D. Plotkin. A per model of polymorphism and recursive types. In *Proceedings of LICS*, pages 355–365, 1990.

[2] A. Ahmed. *Semantics of Types for Mutable State*. PhD thesis, Princeton University, 2004.

[3] A. Ahmed. Step-indexed syntactic logical relations for recursive and quantified types. In *Proceedings of ESOP*, pages 69–83, 2006.

[4] A. Ahmed, A. Appel, and R. Virga. A stratified semantics of general references. In *Proceedings of LICS*, pages 75–84, 2002.

[5] A. Ahmed, M. Fluet, and G. Morrisett. L3: A linear language with locations. *Fundam. Inf.*, 77(4):397–449, 2007.

[6] A. Ahmed, D. Dreyer, and A. Rossberg. State-dependent representation independence. In *Proceedings of POPL*, pages 340–353, 2009.

[7] R. M. Amadio. Recursion over realizability structures. *Information and Computation*, 91(1):55–85, 1991.

[8] R. M. Amadio and P.-L. Curien. *Domains and Lambda-Calculi*. Cambridge University Press, 1998.

[9] P. America and J. J. M. M. Rutten. Solving reflexive domain equations in a category of complete metric spaces. *J. Comput. Syst. Sci.*, 39(3):343–375, 1989.

[10] A. Appel, R. Dockins, and A. Hobor. Mechanized semantic library. `http://msl.cs.princeton.edu/`, 2009.

[11] A. W. Appel and D. A. McAllester. An indexed model of recursive types for foundational proof-carrying code. *ACM Trans. Program. Lang. Syst.*, 23(5): 657–683, 2001.

[12] A. W. Appel, P. Melliès, C. D. Richards, and J. Vouillon. A very modal model of a modern, major, general type system. In *Proceedings of POPL*, pages 109–122, 2007.

[13] N. Benton and C.-K. Hur. Biorthogonality, step-indexing and compiler correctness. In *Proceedings of ICFP*, pages 97–108, 2009.

[14] N. Benton and B. Leperchey. Relational reasoning in a nominal semantics for storage. In *Proceedings of TLCA*, pages 86–101, 2005.

[15] N. Benton, L. Beringer, M. Hofmann, and A. Kennedy. Relational semantics for effect-based program transformations: Higher-order store. In *Proceedings of PPDP*, pages 301–312, 2009.

[16] N. Benton, A. Kennedy, C. Varming, and L. Birkedal. Formalizing domains, ultrametric spaces and semantics of programming languages. Manuscript. Available at `http://www.itu.dk/people /birkedal/papers/formalizing-semantics.pdf`, 2010.

[17] B. Biering, L. Birkedal, and N. Torp-Smith. Bi-hyperdoctrines, higher-order separation logic, and abstraction. *ACM Trans. Program. Lang. Syst.*, 29(5), 2007.

[18] L. Birkedal and R. W. Harper. Constructing interpretations of recursive types in an operational setting. *Information and Computation*, 155:3–63, 1999.

[19] L. Birkedal, N. Torp-Smith, and H. Yang. Semantics of separation-logic typing and higher-order frame rules for Algol-like languages. *LMCS*, 2(5:1), 2006.

[20] L. Birkedal, K. Støvring, and J. Thamsborg. The category-theoretic solution of recursive metric-space quations. Technical Report ITU-2009-119, IT University of Copenhagen, 2009.

[21] L. Birkedal, K. Støvring, and J. Thamsborg. Realizability semantics of parametric polymorphism, general references, and recursive types. In *Proceedings of FOSSACS*, pages 456–470, 2009.

[22] N. Bohr and L. Birkedal. Relational reasoning for recursive types and references. In *Proceedings of APLAS*, pages 79–96, 2006.

[23] F. Cardone. Relational semantics for recursive types and bounded quantification. In *Proceedings of ICALP*, pages 164–178, 1989.

[24] A. Charguéraud and F. Pottier. Functional translation of a calculus of capabilities. In *Proceedings of ICFP*, pages 213–224, 2008.

[25] K. Crary and R. Harper. Syntactic logical relations for polymorphic and recursive types. *Electronic Notes in Theoretical Computer Science*, 172:259–299, 2007.

[26] K. Crary, D. Walker, and G. Morrisett. Typed memory management in a calculus of capabilities. In *Proceedings of POPL*, pages 262–275, 1999.

[27] D. Dreyer, G. Neis, A. Rossberg, and L. Birkedal. A relational modal logic for higher-order stateful ADTs. In *Proceedings of POPL*, pages 185–198, 2010.

[28] A. Hobor. *Oracle Semantics*. PhD thesis, Princeton University, 2008.

[29] A. Hobor, A. Appel, and F. Nardelli. Oracle semantics for concurrent separation logic. In *Proceedings of ESOP*, pages 353–367, 2008.

[30] A. Hobor, R. Dockins, and A. Appel. A theory of indirection via approximation. In *Proceedings of POPL*, pages 171–184, 2010.

[31] P. Johann and J. Voigtländer. A family of syntactic logical relations for the semantics of Haskell-like languages. *Information and Computation*, 207(2): 341–368, 2009.

[32] P. B. Levy. Possible world semantics for general storage in call-by-value. In *Proceedings of CSL*, pages 232–246, 2002.

[33] D. B. MacQueen, G. D. Plotkin, and R. Sethi. An ideal model for recursive polymorphic types. *Information and Control*, 71(1/2):95–130, 1986.

[34] P.-A. Melliès and J. Vouillon. Recursive polymorphic types and parametricity in an operational framework. In *Proceedings of LICS*, pages 82–91, 2005.

[35] A. Nanevski, G. Morrisett, and L. Birkedal. Polymorphism and separation in hoare type theory. In *Proceedings of ICFP*, pages 62–73, 2006.

[36] B. C. Pierce. *Types and Programming Languages*. The MIT Press, 2002.

[37] A. M. Pitts and I. D. B. Stark. Operational reasoning for functions with local state. In A. D. Gordon and A. M. Pitts, editors, *Higher-Order Operational Techniques in Semantics*, Publications of the Newton Institute, pages 227–273. Cambridge University Press, 1998.

[38] F. Pottier. Generalizing the higher-order frame and anti-frame rules. Unpublished, July 2009.

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

[40] U. S. Reddy and H. Yang. Correctness of data representations involving heap data structures. *Science of Computer Programming*, 50(1–3):129–160, March 2004.

[41] B. Reus and J. Schwinghammer. Separation logic for higher-order store. In *Proceedings of CSL*, pages 575–590, 2006.

[42] B. Reus and T. Streicher. Semantics and logic of object calculi. In *Proceedings of LICS*, pages 113–124, 2002.

[43] J. C. Reynolds. Separation logic: A logic for shared mutable data structures. In *Proceedings of LICS*, pages 55–74, 2002.

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

[45] J. Schwinghammer, L. Birkedal, and K. Støvring. A step-indexed Kripke model of hidden state via recursive properties on recursively defined metric spaces. Manuscript. Available at `http://www.itu.dk/people/birkedal/papers/relpoms-antiframe-conf.pdf`, 2010.

[46] J. Schwinghammer, H. Yang, L. Birkedal, F. Pottier, and B. Reus. A semantic foundation for hidden state. In *Proceedings of FOSSACS*, pages 2–17, 2010.

[47] M. B. Smyth. Topology. In S. Abramsky, D. Gabbay, and T. S. E. Maibaum, editors, *Handbook of Logic in Computer Science*. Oxford University Press, 1992.

[48] I. Stark. Categorical models for local names. *LISP and Symbolic Computation*, 9(1):77–107, Feb. 1996.

[49] E. Sumii. A complete characterization of observational equivalence in polymorphic lambda-calculus with general references. In *Proceedings of CSL*, pages 455–469, 2009.

[50] J. Vouillon and P.-A. Melliès. Semantic types: a fresh look at the ideal model for types. In *Proceedings of POPL*, pages 52–63, 2004.