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-sb.de

Kristian Støvring  
IT University of Copenhagen, Denmark  
kss@itu.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.

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 [13, 19, 25, 27, 33, 34]. 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 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, 5, 10, 11], 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 world equations is crucial in some applications, such as the semantics of various higher-order frame and anti-frame rules [36, 37]. 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 [18, 36, 37]. 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, 5, 10, 11]. Unlike these step-indexed models, which only provide solutions to approximated versions of recursive equations, our approach provides solutions to the equations proper. In the paper, we demonstrate the benefits of our method by presenting the first semantic model of Charguéraud and Pottier’s capability calculus [21]. This calculus is a substructural type system for a higher-order ML-like language with state, and imposes a non-trivial soundness issue, because a model needs to involve a recursively defined operation on a recursively-defined set of worlds. 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.\footnote{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 one [36]. Please see the technical appendix for this model.}

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 [26], 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, 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. [26], 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.

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 but point out that we use τ for values, e for expressions and τ for types, in particular e[τ] 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 [12, 27, 32, 39], 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 ∈ W is a finite map from locations (modeled 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:

\[ V = \text{set of values, including locations} \]
\[ W = \mathbb{N} \rightarrow \text{max } T \]
\[ T = W \rightarrow \text{Pred}(V) \]

Note that in the equation for T, we impose the 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 ref τ can be defined roughly as

\[ \text{ref } \tau \in (W, \tau), \]

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.

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 the cardinality reason, there are no solutions to the above equations in the category of sets; unfolding the above equations we get \( W = \mathbb{N} \rightarrow_{\text{max}} (W \rightarrow \text{Pred}(V)) \) 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. [26], 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 applications of interest.

In this paper, we follow a different approach, which involves finding an appropriate simple category of metric spaces in which to solve the original recursive equations. This technique has been developed in the setting of denotational semantics and domain theory. We show that this approach can also be applied to the setting of operational semantics and step indexing. In the next subsections, we explain this by giving a Kripke model of ML references, first using domain-theoretic 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. A 1-bounded ultrametric space \( (X, d) \) is a metric space where the distance function \( d : X \times X \rightarrow \mathbb{R} \) 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 \( f : X_1 \rightarrow X_2 \) 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 \( \text{CBUlt}_{\text{nc}} \). Products are given by the Cartesian product where the distance is the maximum of the componentwise distances. The exponential \( (X_1, d_1) \rightarrow (X_2, d_2) \) has the set of non-expansive functions from \( (X_1, d_1) \) to \( (X_2, d_2) \) as underlying set, and distance function: \( d_{X_1 \rightarrow X_2}(f, g) = \sup\{d_2(f(x), g(x)) \mid x \in X_1\} \). For any set S and space \( (X, d) \in \text{CBUlt}_{\text{nc}} \), the set of finite partial functions \( S \rightarrow_{\text{fin}} X \) from S to X is again a complete, 1-bounded ultrametric space with distance function given by \( d(f, g) = 1 \), if the domain of f and g are not equal, and \( d(f, g) = \max\{d(f(s), g(s)) \mid s \in \text{dom}(f)\} \), if the domain of f and g are equal.

A functor \( F : \text{CBUlt}_{\text{nc}}^\text{op} \times \text{CBUlt}_{\text{nc}} \rightarrow \text{CBUlt}_{\text{nc}} \) 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 some \( \delta < 1 \) 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 \( d'(x, y) = \delta \cdot d(x, y) \). By shrinking, a locally non-expansive functor \( F \) yields a locally contractive functor \( \delta \cdot F \). For a less condensed introduction to ultrametric spaces we refer to [38].

It is well-known that one can solve recursive domain equations in \( \text{CBUlt}_{\text{nc}} \) by an adaptation of the inverse-limit method from classical domain theory:

**Theorem 2.1** (America-Rutten [8]). Let \( F : \text{CBUlt}_{\text{nc}}^\text{op} \times \text{CBUlt}_{\text{nc}} \rightarrow \text{CBUlt}_{\text{nc}} \) be a locally contractive functor. Then there exists a unique (up to isomorphism) \( (X, d) \in \text{CBUlt}_{\text{nc}} \) 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 conventional when working with bisected metric spaces: in such a space, \( x \equiv y \) means that \( d(x, y) \leq 2^{-n} \). We use two facts on \( \equiv \). First, each relation \( \equiv \) is an equivalence relation because of the ultrametric inequality. We are therefore justified in referring to the relation \( \equiv \) as "\( n \)-equality." Second,

2010/4/6
the distance of a bisected metric space is bounded by 1. In other words, the relation \( x \leq y \) always holds.

**Proposition 2.3.** Let \((X_1, d_1)\) and \((X_2, d_2)\) be bisected metric spaces. A function \( f : X_1 \rightarrow X_2 \) is non-expansive if and only if \( x_1 \leq x_2 \Rightarrow d(f(x_1), f(x_2)) \leq d(x_1, x_2) \) holds for all \( x_1, x_2 \in X_1 \) and all natural numbers \( n > 0 \).

### 2.2 General Recipe and Domain-Theoretic Model

We follow now the idea outlined earlier and reformulate the recursive equations (1) in CBUlt to find solutions within this category. Concretely, the proposal suggests to use the following recipe:

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 \( \mathit{Pred}(V) \) in CBUlt. Elements in this object represent predicates on values.
3. Solve a recursive domain equation below in CBUlt:

   \[
   \hat{T} \cong \frac{1}{2} \otimes \left( (\mathbb{N} \to_J \hat{T}) \to_{\mathit{mon}} \mathit{Pred}(V) \right).
   \]

   \( (2) \)

4. Define \( T \) and \( W \) using \( \hat{T} \):

   \[
   W = \mathbb{N} \to_{\mathit{fin}} \hat{T}, \quad T = W \to_{\mathit{mon}} \mathit{Pred}(V).
   \]

   \( (3) \)

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 \( \mathbb{N} \to_{\mathit{fin}} \hat{T} \): For \( w, w' \in \mathbb{N} \to_{\mathit{fin}} \hat{T} \), we have \( w \leq w' \) 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 [8]. The equivalence is well-formed in CBUlt, and it has a unique solution up to isomorphism by Theorem 2.1.

The recipe has been used by Birkedal, Støvring and Thamsborg [18], when they gave a relationally-parametric domain-theoretic model of a call-by-value language with impredicative borg [18], where they gave a relationally-parametric domain-theoretic model of a call-by-value language with impredicative borg [18], when they gave a relationally-parametric domain-theoretic model of a call-by-value language with impredicative borg [18], when they gave a relationally-parametric domain-theoretic model of a call-by-value language with impredicative borg [18].

**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 \( \mathit{Val} \) 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 \( \mathit{UPred}(\mathit{Val}) \) of subsets of \( \mathbb{N} \times \mathit{Val} \) that are downwards closed in the first \( \mathbb{N} \) component:

\[
\mathit{UPred}(\mathit{Val}) = \{ p \subseteq \mathbb{N} \times \mathit{Val} \mid \forall (k,v) \in p. \forall j < k. (j,v) \in p \}.
\]

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

The idea of considering predicates on step-approximated values is from step-indexed models [2, 5, 10, 11]. Here we go “a step further” and show that the collection \( \mathit{UPred}(\mathit{Val}) \) of such predicates can always be made an object in CBUlt. To do this, for \( p \in \mathit{UPred}(\mathit{Val}) \) 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 \( \mathit{UPred}(\mathit{Val}) \), 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.** \( \{ \mathit{UPred}(\mathit{Val}), d \} \) is a well-defined object in CBUlt.

In fact, the construction in \( \mathit{UPred}(\mathit{Val}) \) does not depend on our choice of \( \mathit{Val} \), and can be applied to any set \( X \), giving a metric space \( \mathit{UPred}(X) \) in CBUlt.

Note that because of this lemma, we can consider uniform predicates \( p \in \mathit{UPred}(X) \) on any sets \( X \).

Hence, the recipe in Section 2.2 is applicable for \( \mathit{Val} \) and \( \mathit{UPred}(\mathit{Val}) \), 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 CBUlt, 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. [24], except that we do not consider recursive types and we split the context for type variables and term variables in two. Term judgments take the form

\[
\Pi; \Gamma; \Sigma \vdash M : \tau
\]

where \( \Pi \) is a context of type variables \( \alpha_1, \ldots, \alpha_n \); \( \Gamma \) is a context of typed term variables \( x_1 : \tau_1, \ldots, x_m : \tau_m \); and \( \Sigma \) 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 an non-expansive function

\[
[\Pi \vdash \tau] : T^{[\Pi]} \rightarrow T
\]

in CBUlt. The definition is shown in Figure 1. In the figure, we use \( \eta \) for environments for \( \Xi \), i.e., elements in the product space \( T^{[\Xi]} \) in CBUlt. Notice that in the case for \( \Xi \vdash \rho \tau \), we use \( \rho \)-equality in the space \( T \) and that \( T^{[\Xi \vdash \tau]} \) generalizes \( [\Xi \vdash \rho \tau] \) from values to expressions.

**Lemma 2.5.** \( [\Xi \vdash \tau] \) and \( T^{[\Xi \vdash \tau]} \) are well-defined. In particular, in a category of pre-ordered ultrametric spaces [17]. The latter technique is more general, but for this paper we do not need such pre-ordered spaces.
The operational semantics is defined between configurations $(t \mid h)$ for the union of two such heaps.

The operational semantics is defined between configurations $(t \mid h)$ for the union of two such heaps. 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 [10]. 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. can be obtained by a specialization of our general approach. In Section 4.2, we will also highlight the advantages of using metric spaces.

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 [3, 21, 22]. In this section, we give a step-indexed model for a substantial fragment of Charguéraud and Pottier’s capability calculus [21]. Our model provides an alternative soundness proof to the translation and progress and preservation results in [21, 31], and allows for the analysis of soundness of extensions. We illustrate this latter point by considering an extension of the language with higher-order frame rules [15, 36], 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 sections. 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.

3.1 A Calculus of Capabilities

In the following presentation, we keep close to the notation of Charguéraud and Pottier [21, 31]. 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 to describe heap properties (much like the region quantifier in [31], and allows for the analysis of soundness of extensions. We illustrate this latter point by considering an extension of the language with higher-order frame rules [15, 36], 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 sections. 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.

Figure 4. Interpretation of contexts and well-typed expressions

- for all $\eta \in T^{\Xi}$, both $[\Xi \vdash \tau]_{\eta}$ and $\mathcal{E}[\Xi \vdash \tau]_{\eta}$ are non-expansive and monotonous; and
- $[\Xi \vdash \tau]$ and $\mathcal{E}[\Xi \vdash \tau]$ are non-expansive maps on $\eta$’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 $\Xi; \Gamma; \Sigma \vdash t \downarrow^{\text{w}} \tau$, where $\downarrow^{\text{w}}$ stands for the recursive procedure $f$ with body $t$. If $f$ does not appear in $t$, we may simply write it as $\lambda x. t$.

The operational semantics is defined between configurations $(t \mid h)$ of the closed expression $t$ and a heap $h$. In the previous section, a heap $h$ is a finite map from locations to closed syntactic values. Also, we remind the reader of our notation $(t \mid h)$ 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 $\tau$ classify values, and memory types $\theta$ (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
Variables
\[ \xi ::= \alpha \mid \beta \mid \gamma \mid \sigma \]

Capabilities
\[ C ::= C \otimes C \mid 0 \mid C \wedge C \mid \{ \sigma : \theta \} \mid \exists \sigma.C \mid \gamma \mid \mu \gamma.C \mid \forall \xi.C \]

Value types
\[ \tau ::= \tau \otimes \tau \mid 0 \mid 1 \mid \text{int} \mid \tau + \tau \mid \tau \times \tau \mid \chi \rightarrow \chi \mid \sigma \mid \alpha \mid \mu \alpha.\tau \mid \forall \xi.\tau \]

Memory types
\[ \theta ::= \theta \otimes \tau \mid \tau + \theta \mid \theta \times \theta \mid \text{ref } \theta \mid \theta \wedge C \mid \exists \theta.C \mid \beta \mid \mu \beta.\theta \mid \forall \xi.\theta \]

Computation types
\[ \chi ::= \chi \otimes \tau \mid \chi \wedge C \mid \exists \chi \]

Value environments
\[ \Delta ::= \Delta \otimes C \mid \emptyset \mid \Delta, x : \tau \]

Linear environments
\[ \Gamma ::= \Gamma \otimes C \mid \emptyset \mid \Gamma, x : \chi \mid \Gamma \mid C \]

Figure 5. Capabilities and types

\[ v ::= x \mid () \mid \text{inj}_1 v \mid (v_1, v_2) \mid \text{proj}_1 v \mid \text{ref } v \mid \text{get } v \mid \text{set } v \]

Figure 3. Syntax of values and expressions

\[ (\mu f.\lambda x . t) \leftarrow i \mid f := \mu f.\lambda x . t, \ x := i \mid h \]

\[ \text{proj}^i (v_1, v_2) \leftarrow i \mid v_1 \mid h \quad \text{for } i = 1, 2 \]

\[ \text{case}(v_1, v_2, \text{inj}_i) \leftarrow i \mid v_1 \mid v_2 \mid h \quad \text{for } i = 1, 2 \]

\[ \text{ref } v \leftarrow h \mid \text{get } h \mid \text{set } h \]

\[ \text{proj}_1 (v) \leftarrow () \mid (\text{ref } v \mid h) \quad \text{if } h \notin \text{dom } h \]

\[ \text{proj}_1 (v) \leftarrow () \mid (\text{get } h) \quad \text{if } h \notin \text{dom } h \]

\[ \text{proj}_1 (v) \leftarrow () \mid (\text{set } h \mid v, t) \quad \text{if } h \notin \text{dom } h \]

Figure 4. Operational semantics

types are linear, and correspondingly there is a distinction between value type environments and the more general linear environments.

A region \( \sigma \) 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 \( \gamma \) and permit recursively defined capabilities. A singleton capability \( \{ \sigma : \theta \} \) asserts that the value denoted by \( \sigma \) has type \( \theta \), and moreover it represents the ownership of both this value and the fragment of the heap described by \( \theta \). Thus, it is similar to the points-to predicate of separation logic: for example, the capability \( \{ \sigma : \text{ref } \tau \} \) means that \( \sigma \) denotes the address of a reference cell, and that the “owned” part of the heap stores a value of type \( \tau \) 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 \( \ast \)-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 \( \sigma \ast \{ \sigma : \text{ref } \sigma \} \rightarrow \sigma \ast \{ \sigma : \text{ref } \sigma \} \) 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 \( \gamma \) for \( \mu \gamma.C \) to be well-formed. By this we mean that the recursion must go through one of the type constructors \( +, \times, \rightarrow \) 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 connects \( C \) 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 [15]. 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. 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 \( \Gamma \) serves as a precondition and \( \chi \) 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 \( \Gamma \) and \( \chi \), via \( \ast \cdot \otimes C \). As with the higher-order frame rules in separation logic, this is useful for reasoning about information hiding [15].

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 \( \text{UPred}(X) \) of uniform predicates on a set \( X \) is defined by

\[ \text{UPred}(X) = \{ p \subseteq N \times X \mid \forall k,v \in p. \forall j \leq k. (j,v) \in p \} \]

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

---

Note that (13) and (14) let us move capabilities between assumptions – a form of ownership transfer.
the set of all upwards closed and uniform predicates on \( A \), and we define
\[
p_{[k]} = \{(j, a) \in p | j < k\}.
\]
As in Section 2.3 on \( \text{UPred}(V) \), 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 \( \text{UPred}^1(A) \) as well. We equip \( \text{UPred}^1(A) \) with the same distance function \( d \) as \( \text{UPred}(A) \) in Section 2.3. This makes \( (\text{UPred}^1(A), d) \) an object of \( \text{CBU}_{\text{loc}} \).

In our model, we use \( \text{UPred}^1(A) \) with the following concrete instances for the partial order \( (A, \sqsubseteq) \):
- \((\text{Heap}, \sqsubseteq)\) where \( h \sqsubseteq h' \) iff \( h = h' \) or for some \( h_0 \# h\),
- \((\text{Val}, \sqsubseteq)\) where \( u \sqsubseteq v \) iff \( u = v \),
- \((\text{Val} \times \text{Heap}, \sqsubseteq)\) where \( (u, h) \sqsubseteq (v, h') \) iff \( u = v \) and \( h \sqsubseteq h' \).

We also use variants of the latter two instances where the set \text{Val} is replaced by the set of value substitutions, \( \text{Env} \), and by the set of expressions, \( \text{Exp} \).

On \( \text{UPred}^1(\text{Heap}) \), ordered by subset inclusion, we have a complete Heyting BI algebra structure [14]. Meets and joins are given by set-theoretic intersections and unions, resp., and implication, separating conjunction and separating implication are given by
\[
(k, h) \in p \rightarrow q \iff \forall j \leq k. \forall h' \sqsupseteq h. (j, h') \in q
\]
\[
(k, h) \in p_1 \land p_2 \iff \exists h_1, h_2. h = h_1 \land h_2 \land (k, h_1) \in p_1 \land (k, h_2) \in p_2
\]
\[
(k, h) \in p \rightarrow q \iff \forall j \leq k. \forall h' \sqsupset h. (j, h') \in p \rightarrow (j, h') \in q
\]
The set for \( \ast \) is given by \( I = \mathbb{N} \times \text{Heap} = \mathbb{T} \). Up to the natural number indexing, this is just the standard intuitionistic (in the sense that it is not “tight”) model of separation logic [35].

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 \( W \rightarrow \text{UPred}^1(\text{Heap}) \). Consequently, we solve in \( \text{CBU}_{\text{loc}} \) the following recursive world equation:
\[
W \cong \frac{1}{2} \cdot W \rightarrow \text{UPred}^1(\text{Heap}) \tag{46}
\]
monoids
\[ C_1 \circ C_2 \overset{\text{def}}{=} (C_1 \circ C_2) \times C_2 \] (4)
\[ (C_1 \circ C_2) \circ C_3 = C_1 \circ (C_2 \circ C_3) \] (5)
\[ C \circ \emptyset = C \] (6)
\[ (C_1 \times C_2) \times C_3 = C_1 \times ((C_2 \times C_3) \circ C_2) \] (7)
\[ C \times \emptyset = C \] (8)
\[ C_1 \times C_2 = C_2 \times C_1 \] (9)

monoid actions
\[ (\cdot \times C_1) \times C_2 = (\cdot \times (C_1 \circ C_2)) \times (\cdot \times C_2) \]
\[ (\cdot \times C_1) \times C_2 = (\cdot \times (C_1 \circ C_2)) \times (\cdot \times C_2) \]
action by * on singleton
\[ \{ \sigma : \theta \} \circ C = \{ \sigma : \theta \times C \} \]
action by * on linear environments
\[ (\Gamma, x : \chi) \circ C = \Gamma, x : (\chi \times C) \]
action by * on capabilities, types, and environments
\[ \{ \cdot \times \} \circ C = \{ \cdot \times \} \circ C \]
\[ \{ \exists \sigma, \cdot \} \circ C = \exists \sigma, \cdot \circ C \] (11)
\[ \emptyset \circ C = \emptyset \]
\[ \{ \sigma : \theta \} \circ C = \{ \sigma : \theta \times C \} \]
\[ 0 \circ C = 0 \]
\[ 1 \circ C = 1 \]
\[ \text{int} \circ C = \text{int} \]
\[ (\theta_1 + \theta_2) \circ C = (\theta_1 \circ C) + (\theta_2 \circ C) \]
\[ (\theta_1 \times \theta_2) \circ C = (\theta_1 \times C) \times (\theta_2 \circ C) \]
\[ (\forall \xi, \theta) \circ C = \forall \xi, (\theta \circ C) \] (14)
\[ (\chi_1 \times \chi_2) \circ C = (\chi_1 \circ C) \times (\chi_2 \circ C) \]
\[ \{ \sigma : \theta \} \circ C = \{ \sigma : \theta \times C \} \]
\[ (\Gamma, x : \chi) \circ C = \Gamma, x : \chi \times C \]
\[ (\Gamma \times C_1) \circ C_2 = (\Gamma \times C_2) \times (C_1 \circ C_2) \]

region abstraction
\[ \exists \sigma, \exists \sigma_2, \cdot = \exists \sigma, \exists \sigma_2, \cdot \]
\[ \cdot \times (\exists \sigma, C) = \exists \sigma, (\cdot \times C) \]
\[ \{ \sigma_1 : \exists \sigma_2, \theta \} = \exists \sigma_2, \{ \sigma_1 : \theta \} \] (32)

focusing
\[ \{ \sigma_1 : \text{ref} \theta \} = \exists \sigma_2, \{ \sigma_1 : \text{ref} [\sigma_2] \} \times \{ \sigma_2 : \theta \} \]
\[ \{ \sigma : \theta_1 \times \theta_2 \} = \exists \sigma, \{ \sigma : [\sigma_1] \times [\sigma_2] \} \times \{ \sigma_1, \sigma_2 : \theta_1 \times \theta_2 \} \]
\[ \{ \sigma : \theta_1, \theta_2 \} = \exists \sigma, \{ \sigma : [\sigma_1, \sigma_2 : \theta_1 \times \theta_2] \} \times \{ \sigma_1 : \theta_1 \times \theta_2 \} \]
\[ \{ \sigma : 0 + \theta_2 \} = \exists \sigma_2, \{ \sigma : 0 + [\sigma_2] \} \times \{ \sigma_2 : \theta_2 \} \]

recession
\[ \mu_\gamma, C = C[\gamma = \mu_\gamma, C] \]
\[ \mu_\alpha, \pi = \pi[\alpha = \mu_\alpha, \pi] \]
\[ \mu_\beta, \theta = \theta[\beta = \mu_\beta, \theta] \]

(first-order) frame axiom
\[ \chi_1 \rightarrow \chi_2 \leq (\chi_1 \times C) \rightarrow (\chi_2 \times C) \] (42)
free
\[ C_1 \times C_2 \leq C_1 \] (43)
singleton
\[ \tau \leq \exists \sigma, [\sigma] \times \{ \sigma : \tau \} \] (44)
\[ [\sigma] \times \{ \sigma : \tau \} \leq \tau \times \{ \sigma : \tau \} \] (45)

Figure 6. Structural equivalence

By definition of the n-equality, \( p \times q \equiv p' \times q' \) is equivalent to \( (j, h) \in p \times q \iff (j, h) \in p' \times q' \) for all \( j < n \), which follows easily from the assumptions that \( p \equiv p' \) and \( q \equiv q' \).

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 reflects the syntactic abbreviation \( C_1 \circ C_2 = C_1 \circ (C_2 \circ C_3) \). The semantic domain corresponding to each syntactic category is a set of (contractively world-dependent) upwards closed and uniform predicates:
\[ \text{Val} \]
\[ \text{Val} \]
\[ \text{Val} \times \text{Heap} \]
In particular, in each case there is an action of the BI unit under \( \circ \), as described in Proposition 3.3. Note that \( \text{Cap} = \frac{1}{2} \circ W \times \text{Val} \times \text{Val} \times \text{Heap} \) acts on itself, via the isomorphism \( \iota \) 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 [36, 37]. Moreover, due to the shrinking factor \( \delta = \frac{1}{2} \), this action is contractive in its right-hand side: for every \( p, r \in \text{Cap} \), the assignment \( r \mapsto p \otimes (r) \) is a contractive endomap on Cap. This observation explains why the (syntactic) invariant extension is formally contractive in its right-hand side.

We also consider a further overloading of the separating conjunction. It is the below generalization \( \text{Cap} \in p, r \in \delta \) order frame (and also anti-frame) inference rules and the associated form of contractiveness on non-values:

\[
\text{Cap} \in (k, (a, h \cdot h')) \in S \land (k, h') \in q \land h \neq h'.
\]

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

\[
(S * r)(w) = S(w) * r(w).
\]

This provides a second monoid action, with respect to the monoid structure given by the separating conjunction on \( \text{Cap} \).

**Proposition 3.4 (Monoid and monoid action).** \( (\text{Cap}, *, I) \) 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 \( \text{UPred}^1(A \times \text{Heap}) \), i.e., \( S * I = S = (S + p) * q = S * (p * q) \).

The interpretation of capabilities and types is given in Figure 9. This interpretation depends on an environment \( \eta \), which maps region names \( \sigma \in \text{RegName} \) to closed values \( \eta(\sigma) \in \text{Val} \), capability variables \( \gamma \) to semantic capabilities \( \eta(\gamma) \in \text{Cap} \), and type variables \( \alpha \) and \( \beta \) to semantic types \( \eta(\alpha) \in \text{VT} \) and \( \eta(\beta) \in \text{MT} \). As indicated above, the semantics of capabilities is defined in terms of the BI structure on \( \text{Cap} \). The semantics of memory types uses the action of \( \text{Cap} \) on MT described in (47). It also makes explicit the aliasing information contained in memory types: for instance, the two components of a pair 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 \( \tau \) considered as memory type, \( \| \tau \| \) on the right-hand side refers to the value type interpretation. Note that the computation types \( \chi \) form a subset of the memory types, and thus obtain their interpretation in MT.

Let \( \text{Env} \) be the set of finite maps from variables to closed values. Duplicate environments are heap-independent, and interpreted as non-expansive maps \( \frac{1}{2} \cdot W \rightarrow \text{UPred}^1(\text{Env}) \). Similarly, linear environments are interpreted as non-expansive maps \( \frac{1}{2} \cdot W \rightarrow \text{UPred}^1(\text{Env} \times \text{Heap}) \). 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 \( w \) is simply passed through and the index is decreased (whenever justified by the operational semantics) so as to ensure that the 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 (Expression typing).** Let \( S \in \text{MT} \). Then the function \( \mathcal{E}(S) : W \rightarrow \text{UPred}^1(A \times \text{Heap}) \) is defined by \( (k, (t, h)) \in \mathcal{E}(S)(w) \) if

\[
\forall j \leq k, t', h'. (t \upharpoonright h') \rightarrow j \quad \Rightarrow \quad (k - j, (t', h')) \in S(w) * \mathcal{E}^{-1}(w)(w)(\text{emp}).
\]

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 \( S_1, S_2 \in \text{MT} \), expressions \( t \) and \( h \in \text{Heap} \), if \( w_1 \equiv w_2 \) in \( W \), \( S_1 \equiv S_2 \) and \( t \not\in \text{Val} \), then for all \( k \leq n \),

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

**Proof.** Let \( w_1 \not\equiv w_2 \), and observe that this implies \( S_1(w_1) \not\equiv S_2(w_2) \) and \( \mathcal{E}^{-1}(w_1)(\text{emp}) \not\equiv \mathcal{E}^{-1}(w_2)(\text{emp}) \) for any \( m < n \). Now assume that \( (k, (t, h)) \in \mathcal{E}(S_1)(w_1) \) for some \( k \leq n \). We must show that \( (k, (t, h)) \in \mathcal{E}(S_2)(w_1) \). For this, suppose that \( (t \upharpoonright h') \rightarrow j \) for some \( j < k \) where \( (t' \upharpoonright h') \) is irreducible. The assumption \( (k, (t, h)) \in \mathcal{E}(S_1)(w_1) \) yields \( (k - j, (t', h')) \in S_1(w_1) * \mathcal{E}^{-1}(w_1)(\text{emp}) \). In particular, \( t' \in \text{Val} \) and therefore \( t \neq t' \) by the assumption that \( t' \not\in \text{Val} \). Thus we must have \( j > 0 \), and therefore \( k - j < k \leq n \) which by the above observations means that \( (k - j, (t', h')) \in S_2(w_2) * \mathcal{E}^{-1}(w_2)(\text{emp}) \).

The direction from right to left is symmetric. \( \square \)

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 \( \mathcal{E}^{-1}(w)(\text{emp}) \) on heaps, which is conjuncted 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 \( E \). Second, by additionally adjoining \( r \) as we inake the first-order frame property. Finally, the quantification over indices \( j < k \) achieves that \( [x_1 \rightarrow x_2] \), \( w \) is in \( \text{UPred}^1(\text{Val}) \). There are two explanations why we require that \( j \) be strictly less than \( k \) in the definition of \( [x_1 \rightarrow x_2] \). Technically, the use of \( \mathcal{E}^{-1}(w) \) in the definition “undoes” the scaling by \( \frac{1}{2} \), and the strictly smaller index is needed to ensure the non-expansiveness of \( [x_1 \rightarrow x_2] \) as a function \( \frac{1}{2} \cdot W \rightarrow \text{UPred}^1(\text{Val}) \). 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.

**Proposition 3.7.** The interpretation in Figure 9 is well-defined: all the \( \frac{1}{2} \cdot \)’s map into the declared sets, and the recursive definitions of capabilities and types have unique solutions.

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

\[
d(\eta, \eta') = \sup_\xi d(\eta(\xi), \eta'(\xi)).
\]

We then show by simultaneous induction on \( C, \tau, \) and \( \theta \), the following properties:

1. \( [C]_w, [\tau]_w, \) and \( [\theta]_w \) are upwards closed and uniform predicates;
2. \( [C]_w, [\tau]_w, \) and \( [\theta]_w \) are non-expansive functions of \( \eta \) (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 \( \xi \) then \( [C]_\eta[\xi_{\equiv \eta}] \) is contractive; and
4. if \( \theta \) is formally contractive in \( \xi \) then \( [\theta]_\eta[\xi_{\equiv \eta}] \) 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. \( \square \)
Figure 9. Interpretation

This interpretation respects the structural equivalence, i.e., whenever $C_1$ and $C_2$ are equivalent capabilities then $[C_1] = [C_2]$ (and similarly for value and memory types). The proofs of these facts are easy consequences of the definition of $[C]$ and Propositions 3.3 and 3.4, and are given in Appendix C.1. Moreover, the interpretation validates the subtyping axioms, i.e., whenever $\theta_1 \leq \theta_2$ then $[\theta_1] = [\theta_2]$holds for all $\eta$ and $w$. These proofs can be found in Appendix C.2.

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

$$\Delta \vdash v : \tau \quad \Gamma \vdash t : \chi$$

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

$$\models (\Delta \vdash v : \tau) \iff \forall \eta, \forall w \in W, \forall k \in \mathbb{N}, \forall \rho \in [\Delta]_\eta w, (k, \rho(v)) \in [\tau]_\eta w \ .$$

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

$$\models (\Gamma \vdash t : \chi) \iff \forall \eta, \forall w \in W, \forall \kappa \in \mathbb{N}, \forall r \in Cap.$$
I = \omega \times \{[[\_]]\}. However, this complete collection property is not only destroyed by the axiom C_1 \ast C_2 \leq C_1 but also by the anti-frame rule, which we do not consider in this paper though.

**Unique Solutions Proof Principle** In practice, one may have to show type equivalences. We have 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.

### 4. Specialization to Indirection Theory

Hobor, Dockins and Appel [26] 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 \( F : \text{Set} \rightarrow \text{Set} \), and this functor must in a suitable sense have an extension to \( \text{CBUlt} \) in order for our approach to apply. Fortunately, this condition holds for functors on \( \text{Set} \) built with standard constructors. In return for requiring this extra condition, we can obtain an approximate solution that improves on the one constructed in Hobor et al.: the metric-space setup guarantees that all the predicates we consider are so-called hereditary.

We now sketch how the specialization to indirection theory proceeds. The full story, including proofs, can be found in Appendix A.

#### 4.1 Indirection Theory

Assume that we are given a functor \( F : \text{Set} \rightarrow \text{Set} \) and a non-empty set \( O \). Let \( \mathbb{2} = \{0, 1\} \) be the set of “truth values.” Indirection theory begins from the desire to solve the equation

\[
K \cong F(K \times O \rightarrow 2)
\]

in \( \text{Set} \), which is often impossible for the cardinality reason. Instead, one obtains an approximate solution

\[
K \overset{\text{unsquash}}{\underset{\text{squash}}{\rightarrow}} \mathbb{N} \times F(K \times O \rightarrow 2)
\]

consisting of a set \( K \) and functions \( \text{squash} \) and \( \text{unsquash} \) satisfying:

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

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

\[
\text{approx}_m(\psi)(k, o) = (\psi(k, o) \land \text{level}(k) < m).
\]

The idea is that elements of \( K \) have “levels,” and that the function \( \text{approx}_m \) 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 unsquash, but in general not a right inverse:

6Unlike 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.

Unsquash \((\text{squash}(m, \nu)) \) is in some sense an approximation of \((m, \nu)\).

#### 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 \( \text{CBUlt} \). We now assume that the functor \( \hat{F} : \text{Set} \rightarrow \text{Set} \) considered above has a so-called plain lift \( \hat{F} : \text{CBUlt} \rightarrow \text{CBUlt} \). 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 shown in Appendix A. As noted above, plain lifts exist for all the standard constructors (see Proposition A.7). In particular we have plain lifts of the functors of all the examples of Hobor et al.

From Theorem 2.1 and Lemma 2.4, we easily obtain:

**Theorem 4.1.** There is a non-empty, complete, 1-bounded ultra-metric space \( X \) and an isomorphism

\[
\Phi : X \cong \hat{F}\left(\frac{1}{2}[X \rightarrow \text{UPred}(O)]\right),
\]

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 \( \text{UPred}(O) \), since hereditary predicates are, intuitively, “closed under approximation” in the \( K \) component. As mentioned above, we regard this difference as an improvement. Indeed, Hobor et al. state a clear desire to consider hereditary predicates only (Section 5.3) and briefly mention an alternative, more complicated construction of approximate solutions that guarantees that all predicates are hereditary (Section 10). Here we obtain such a guarantee directly from the metric-space setup.

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

\[
K \overset{\text{unsquash}}{\underset{\text{squash}}{\rightarrow}} \mathbb{N} \times \hat{F}(K \times O \rightarrow \text{her} 2)
\]

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 goes even if we disregard recursively defined worlds. Consider the interpretation of recursive types in Section 3. The idea of ‘stepping one down’ when interpreting recursive types seems natural to anyone familiar with step-indexed models. But coming up with the correct criteria on the interpretation function for this to work out properly, also with nested recursive types, is not so easy a priori. If, however, we employ the metric approach, including Banach’s fixed-point theorem, then writing down the requirements as done in

\footnote{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.}
the section is straightforward. Another example is the $\otimes$ operator in the same section, which is 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. [4] or with the indirectness theory of Hobor et al. [26]. But the precise course of action is much less immediate.

Secondly, in comparison with the indirectness theory [26], 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 [26]. In the model provided by indirectness theory, types are arbitrary maps from worlds to values, modulo currying and nomenclature. But, as argued in [26, Section 5.1], we really want types that are both hereditary and monotone. In [26, 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 recognized in the last paragraph of [26, Section 10] where an alternative, and less straightforward, model with only hereditary types in the worlds is sketched. But that means starting the model construction all over from scratch and does not buy us monotonicity. On the other hand, to obtain hereditary types with the metric approach we just use the downwards-closure condition on $UPred(V)$, verify Lemma 2.4 and apply Theorem 2.1. And to work with monotone types, we can apply a slightly stronger existence result [17, Proposition 5.4] for pre-ordered metric spaces. A similar argument goes for the extension to mixed variance functors discussed in [26, Section 10]: it is already supported by the metric-space approach. Indeed, in unpublished work we have used mixed-variance functors to verify that the metric-space approach scales to the elaborate worlds of [4].

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., [13, 18]), and on step-indexed models (e.g., [4]). For such relational reasoning, the worlds are typically more sophisticated than the worlds we have discussed so far. This is because in this case, 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 [4] 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. [24] have shown how to extend the relational step-indexed model [4] 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 [29] 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 [9].

Thus, one may wonder whether our proposed metric approach hinders formalizations. It does not. Following the treatment in [17], Varming and Birkefeld have recently formalized the solutions of recursive metric-space equations in Coq [40] and the step-indexed model of ML references from Section 2.3.

Capabilities In [3], 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, instrumenting a semantic analysis of what they really should denote. Moreover, they did not consider non-trivial combinations of capabilities such as $C_1 \otimes C_2$ and did not include frame rules, etc. The step-indexed model in this paper does not alter the operational semantics, interprets capabilities including $C_1 \otimes C_2$ 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 [21] with a semantic model of their purely functional target calculus. The functional translation in [21] 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 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.

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 in 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 [21], where solving an original recursive equation over worlds played a 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 [26] 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

A. Specialization to Indirection Theory

Faced with a higher-order store recursive equation, Hobor, Dockins and Appel [26] provide an approximative solution. This is a section-retraction pair characterized by the two axioms of indirection theory that elegantly capture the approximative nature of the solution. Our approach is different, we solve the recursion proper in a certain category of metric spaces. In both cases, however, the solution provides a notion of worlds\(^8\) to be used in Kripke models as exemplified already in loc.cit. and in the previous section. In this section we shall argue that our approach is the more general in the sense that, for the same recursive equation, one may build the approximative solution of Hobor et al. from our solution – this is Theorem A.6. A consequence is that, somewhat indirectly, we have shown our method applicable to all examples considered by Hobor et al.

This specialization to indirection theory is not unconditional. The construction presented by Hobor et al. is parameterized over a set-theoretic functor \(F: \text{Set} \to \text{Set}\) but our approach deals in metric-space equations phrased in terms of a locally non-expansive functor on \(\text{CBUlt}\). So we must have one of the latter corresponding to the former or, more precisely, we must have a plain lift of \(F: \text{Set} \to \text{Set}\), as defined below, to apply the specialization. Fortunately, in many cases such a lift exists; indeed it always holds for a set-theoretic functor \(\text{Set}\).\(^9\) Theorem A.6. A consequence is that, somewhat indirectly, we have the sense that, for the same recursive equation, one may build the solution. Our approach is different, we solve the recursion proper in a certain category of metric spaces. In both cases, however, the solution is non-shrinking because \((\text{Set} \to \text{CBUlt}) \leq \text{CBUlt} \leq \text{CBUlt}\) is assumed locally contractive contravariant functor on \(\text{CBUlt}\) and so is the composite of the two.

Envision now a functor \(F: \text{Set} \to \text{Set}\), a non-empty set \(O\) of values and a request for a solution to the recursive equation \(\text{Set} \ni x \in \frac{1}{2} (\text{Set} \to \text{UPred}(O))\) is a locally contractive contravariant functor on \(\text{CBUlt}\) and so is the composite of the two.

Here we put a subscript \(\text{ne}\) on \(\to\) and make it explicit that we are using the space of non-expansive functions.

This is an easy consequence of Theorem 2.1: \(\tilde{F}\) is assumed locally non-expansive and the functor \(\frac{1}{2} (\text{Set} \to \text{UPred}(O))\) is a locally contractive contravariant functor on \(\text{CBUlt}\) and so is the composite of the two.

We deviate from indirection theory as introduced by Hobor et al. on two counts: We do not parameterize over the set of truth values but stick to \(2 = \{0, 1\}\); the generalization, while probably technically feasible, appears unmotivated. More importantly, we build a solution that features only hereditary maps from \(K \times O\) to \(2\), see the definition below. This is a direct consequence of the uniformity required of members of \(\text{UPred}(O)\), lifting the latter constraint would most likely remove the former too. But we regard it as a strength, not a shortcoming, as we really would like to stay hereditary all the way and now we know that ‘unsquashing a knot’ does not invalidate this wish – compare with the discussion in the last paragraph of [26, Section 10].

Theorem A.6. Let \(F: \text{Set} \to \text{Set}\) be a functor with a plain lift \(\tilde{F}: \text{CBUlt} \to \text{CBUlt}\). We can, from the isomorphism of Theorem A.5, build a set \(K\), a subset of hereditary maps \(K \times O \to _\text{her} \to \text{Set}\) of the full function space \(K \times O \to 2\) and two maps

\[K \xrightarrow{\text{unsquash}} \text{Set},\]

with the following three properties:

1. \(\text{squash} \circ \text{unsquash} = 1_K\).
2. \((\text{unsquash} \circ \text{squash})(m, \nu) = (m, \nu)\).
3. \(\psi \in K \times O \to 2 \Rightarrow \psi = \square \psi\).

Here the level \(\text{fst} \circ \text{unsquash} : K \to \text{Set}\) and the map approx\(_m\) : \((K \times O \to _\text{her} \to \text{Set}) \to (K \times O \to _\text{her} \to \text{Set})\) is defined, for each \(m \in \mathbb{N}\), by

\[\text{approx}_m(\psi)(k, o) = \psi(k, o) \land \text{level}(k) < m.\]

And for \(\psi \in K \times O \to 2\) we define \(\square \psi \in K \times O \to 2\) by

\[(\square \psi)(k, o) = \forall l \in K. k A^* \mapsto \psi(l, o),\]

where \(A^*\) is the reflexive, transitive closure of the relation \(A\) on \(K\) defined, for all \(k, l \in K\), by

\[k A \mapsto \text{unsquash}(k) = (m + 1, \nu) \land l = \text{squash}(m, \nu).\]

Proposition A.7. There is a plain lift of any functor built from the identity, constant non-empty sets, products and (possibly finite and partial) maps from a constant set.
B. Specialization to Indirection Theory, Three Proofs

Proof of Proposition A.3. It is immediate that any non-expansive ϕ has the stated property. Assume, on the other hand, that we need to show ϕ non-expansive. Let x, y ∈ X, we must show that d(ϕ(x), ϕ(y)) ≤ d(x, y), where d is the metric on X. We may without loss of generality assume that d(x, y) ≠ 0. But then there is m ∈ N with d(x, y) = 2⁻ᵐ in particular we have d(x, y) ≤ 2⁻ᵐ which we usually write x ≈ m y. From the assumption we get that ϕ(x) =ₙₗ ϕ(y), i.e., that d(ϕ(x), ϕ(y)) ≤ 2⁻ᵐ and we are done.

Proof of Theorem A.6. Let X and Φ be the result of applying Theorem A.5 to F. Note initially that X must be bisected. This is by definition the case for UPred(O) and hence any two elements of X →₀₂ UPred(O) have a distance that is the supremum of a nonempty subset of {0} ∪ {2⁻ᵐ | m ∈ N}. But this set is closed under nonempty supremum and so X →₀₂ UPred(O) is bisected too. Both of the functions 2⁻ᵐ(−) and F preserve the property of being bisected, the former by construction and the latter by assumption. And so X, which is isomorphic to 2⁻ᵐ(2(X →₀₂ UPred(O))), must be bisected.

Without further ado, let us plunge into the construction. For every m ∈ N we know that =ₙₗ is an equivalence relation on X, for x ∈ X we denote by [x], the equivalence class containing x. We let K be the sum of all but the first of the sets of equivalence classes:

\[ K = \sum_{m \geq 1} X/ =ₙₗ \]

Furthermore we let K × O →₀₂ 2 consist of the set-theoretic maps ψ : K × O → 2 such that for any (m, [x]ₚₗ) ∈ K, any o ∈ O and any 0 < n < m we have

\[ ψ([m, [x]ₚₗ], o) \Rightarrow ψ([m, [x]ₚₗ], o). \]

To build squash and unsquash we need auxiliary maps:

\[ \frac{1}{2}(X →₀₂ UPred(O)) \xrightarrow{H} K × O →₀₂ 2 \]

defined by

\[ H(ϕ) = λx ∈ X. (m, o) \mid ϕ(m + 1) \in \{0, 1\} \]

respectively by

\[ B(ψ) = λx ∈ X. (m, o) \mid ϕ(m + 1) \in \{0, 1\}. \]

These are well-defined. To verify this for H take ϕ ∈ 2⁻ᵐ(2(X →₀₂ UPred(O))), (m, [x]ₚₗ) ∈ K and o ∈ O. Notice initially that the choice of the representative x does not matter for if x =ₚₗ y holds for two x, y ∈ X we have ϕ(x) =ₚₗ ϕ(y) too, in particular (m − 1, o) ∈ ϕ(x) if and only if (m − 1, o) ∈ ϕ(y). To prove H(ϕ) ∈ K × O →₀₂ 2 we furthermore take 0 < n < m and assume that H(ϕ)(m, [x]ₚₗ), o) holds, i.e., that (m − 1, o) ∈ ϕ(x). Proving H(ϕ)(n, [x]ₚₗ), o) comes down to showing (n − 1, o) ∈ ϕ(x) which is true by uniformity of ϕ(x).

To verify that B is well-defined we take ψ ∈ 2⁻ᵐ(2(X →₀₂ UPred(O))). First we take x ∈ X and must prove \( \{(m, o) \mid ψ(m + 1, [x]ₚₗ), o) \} \) uniform. So assume that we have \( n < m \in \mathbb{N} \) and \( o ∈ O \) with \( ψ(m + 1, [x]ₚₗ), o) \). Immediately get \( ψ(n + 1, [x]ₚₗ), o) \). Second we take x, y ∈ X with x =ₚₗ y for some \( m \in \mathbb{N} \), we must show that \( B(ψ)(x) =ₚₗ B(ψ)(y) \), i.e., that for all \( n < m \in \mathbb{N} \) and all \( o ∈ O \) we have \( ψ(n + 1, [x]ₚₗ), o) \) iff \( ψ(n + 1, [y]ₚₗ), o) \), but this is immediate since \( [x]ₚₗ = [y]ₚₗ \). Here we used Proposition A.3 to prove non-expansiveness of B(ψ).

Going back and forth with H and B gets you nowhere. For ψ ∈ K × O →₀₂ 2 we get that

\[ H(B(ψ)) = H(λx. (m, o) \mid ψ((m + 1, [x]ₚₗ), o)) = λ(m, [x]ₚₗ), o). \]

and for φ ∈ 2⁻ᵐ(X →₀₂ UPred(O)) we get

\[ B(H(φ)) = B(λx. (m, [x]ₚₗ), o). φ(x) \supseteq (m, 1, o) \]

\[ = λx. (m, o) \mid ϕ(x) \supseteq (m, o) \]

Up until this point, the maps H and B have been merely set-theoretic and not morphisms in CBUlt, indeed, K × O →₀₂ 2 is itself just a set. But now we equip it with the metric induced by the bijection with 2⁻ᵐ(X →₀₂ UPred(O)), i.e., the distance between elements is the distance between the images of these elements under application of B. With this metric we obviously get an object of CBUlt, and the maps H and B are morphisms of CBUlt, indeed, they are isomorphisms. We need this to be able to apply F to them.

Also we need, for each m ∈ N, to define πₘ on 2⁻ᵐ(X →₀₂ UPred(O)) by pointwise application of the restriction map, i.e., for ϕ ∈ 2⁻ᵐ(X →₀₂ UPred(O)) we define

\[ πₘ(ϕ)(x) = ϕ(x)ₚₗ. \]

We should verify that this is a non-expansive map. It has been argued above that 2⁻ᵐ(X →₀₂ UPred(O)) is bisected so by Proposition A.3 we take ϕ₀, ϕ₁ ∈ 2⁻ᵐ(X →₀₂ UPred(O)), n ∈ N, assume ϕ₀ =ₚₗ ϕ₁ and aim to prove πₘ(ϕ₀) =ₚₗ πₘ(ϕ₁). We may without loss of generality assume n > 0. For x ∈ X we get by assumption that ϕ₀(x) =ₚₗ ϕ₁(x) which implies that ϕ₀(0)ₚₗ =ₚₗ ϕ₁(0)ₚₗ too and we are done. Really we would like to talk about the maps (approxₘ)m∈N on K × O →₀₂ 2 but we cannot since squash and unsquash have not been defined yet; instead we deal in (πₘ)m∈N on 2⁻ᵐ(X →₀₂ UPred(O)). We shall need and prove a close correspondence between the two below.

We are now ready to construct the promised set-theoretic maps squash and unsquash. For (m, [x]ₚₗ) ∈ K we define

\[ \text{unsquash}(m, [x]ₚₗ) = (m − 1, λF(H) corners (F×₀₂ H))(x) \]

and for (m, ν) ∈ N × K × O →₀₂ 2 we set

\[ \text{squash}(m, ν) = \left( m + 1, 1 \right. \mid \text{π}_{m+1}(πₘ(\rho))(x) \}

Our first aim to verify is that unsquash is indeed well-defined, i.e., that the choice of the representative x does not matter. For x, y ∈ X with x =ₚₗ y for some m > 0 we get (πₘ)(x) =ₚₗ (πₘ)(y). For any two ϕ₀, ϕ₁ ∈ 2⁻ᵐ(X →₀₂ UPred(O)) we get that if ϕ₀ =ₚₗ ϕ₁ then for any z ∈ X we have ϕ₀(z) =ₚₗ ϕ₁(z). But then

\[ \text{π}_{m-2}(πₘ)(z) =ₚₗ \text{π}_{m-2}(πₘ)(z) \]

so we have πₘ₋₁(πₘ)(z) =ₚₗ πₘ₋₁(πₘ)(z). As F was assumed non-shrinking, we can now conclude that (F×₀₂ H)(πₘ₋₁)(x) =ₚₗ (F×₀₂ H)(πₘ₋₁)(y) and we know that unsquash is well defined.

Before we go on, we need a quick comment on an easily overlooked issue. The maps squash and unsquash are both set-theoretic as desired but really they go between K × N × U(F(K × O →₀₂ 2)) and U × CBUlt → Set is the forgetful functor. But we assumed F a lift of F so

\[ \text{π}_m \circ \text{π}_{m+1} = \text{π}_{m+1} \circ \text{π}_m \]

14

2010/4/6
and the latter is what we usually just write $F(K \times O \to \text{her}_2)$. So the domain respectively codomain of squash and the latter is what we usually just write.

With the issues of well-definedness taken care of, we now pursue the promised equalities. For $(m, [x]_m) \in K$ we calculate as follows:

\[
\begin{align*}
\text{(squash} \circ \text{unsquash})(m, [x]_m) \\
= \text{squash} \left( m - 1, (\hat{F}(H) \circ \hat{F}(\pi_{m-1}) \circ \Phi)(x) \right) \\
= (m, [(\Phi^{-1} \circ \hat{F}(B) \circ \hat{F}(H) \circ \hat{F}(\pi_{m-1}) \circ \Phi)(x)]_m) \\
= (m, (\Phi^{-1} \circ \hat{F}(\pi_{m-1}) \circ \Phi)(x))[m].
\end{align*}
\]

A bit of reasoning remains to show that this is indeed $(m, [x]_m)$. Notice first that we may rewrite

\[
x = (\Phi^{-1} \circ \hat{F}(1_{\hat{F}(X \to \text{unpred}(O))} \circ \Phi)(x).
\]

This means that if we can prove

\[
\pi_{m-1} = m \uparrow (X \to \text{unpred}(O))
\]

then we are done as $\hat{F}$ was assumed locally non-expansive. So take $\varphi \in 1_{\hat{F}(X \to \text{unpred}(O))}$. For any $y \in X$ we get that

\[
\pi_{m-1}(\varphi)(y) = \varphi(y)[m-1] = \varphi(y)
\]

so in $X \to \text{unpred}(O)$ we have $\pi_{n-1}(\varphi) = \varphi$ and we are done because of the shrinking factor.

For $(m, \nu) \in \mathbb{N} \times \hat{F}(K \times O \to \text{her}_2)$ we get

\[
\begin{align*}
\text{(unsquash} \circ \text{squash})(m, \nu) \\
= \text{squash} \left( m + 1, [(\Phi^{-1} \circ \hat{F}(B))(\nu)]_{m+1} \right) \\
= (m, (\hat{F}(H) \circ \hat{F}(\pi_{m}) \circ \Phi \circ \Phi^{-1} \circ \hat{F}(B))(\nu)) \\
= (m, (\hat{F}(H) \circ \hat{F}(\pi_{m}) \circ \hat{F}(B))(\nu)).
\end{align*}
\]

To finish this we need to look into the relationship between $\pi_{m}$ and the map $\text{approx}_{\nu}$. Take $\psi \in K \times O \to \text{her}_2$, we start from one end and get that

\[
\begin{align*}
(\pi_{m} \circ B)(\psi) \\
= \pi_{m}(\lambda x. \{ (n, o) \mid \psi((n + 1, [x]_{n+1}), o) \}) \\
= \lambda x. \{ (n, o) \mid \psi((n + 1, [x]_{n+1}), o) \land n < m \} \\
= \lambda x. \{ (n, o) \mid \text{approx}_{\nu}(\psi)((n + 1, [x]_{n+1}), o) \} \\
= (B \circ \text{approx}_{\nu})(\psi)
\end{align*}
\]

where we remember that level$(n + 1, [x]_{n+1}) = n$ since level is the composite of the first projection and unsquash. Summing up we have proved that

\[
\begin{align*}
\text{(unsquash} \circ \text{squash})(m, \nu) \\
= (m, (\hat{F}(H) \circ \hat{F}(B) \circ \text{approx}_{\nu})(\nu)) \\
= (m, \text{approx}_{\nu}(\nu))
\end{align*}
\]

as desired – again we applied that $\hat{F}$ is a lift of $F$.

We now consider the third property; we need to prove that the subset $K \times O \to \text{her}_2$ of the full function space $K \times O \to 2$ coincides with the functions that are hereditary in the sense that they are fixed under application of $\square$. Take initially $(m, [x]_m)$ and $(n, [y]_n)$ in $K$, we get that $(m, [x]_m) A (n, [y]_n)$ holds iff we have

\[
\text{unsquash}(m, [x]_m) = (l + 1, \nu) \land (n, [y]_n) = \text{squash}(l, \nu)
\]

\[
\Rightarrow (m - 1, (\hat{F}(H) \circ \hat{F}(\pi_{m-1}) \circ \Phi)(x)) = (l + 1, \nu) \land
\]

\[
(n, [y]_n) = (l + 1, [(\Phi^{-1} \circ \hat{F}(B))(\nu)]_{l+1})
\]

\[
\Rightarrow m = n + 1 \land [y]_n = ((\Phi^{-1} \circ \hat{F}(B) \circ \hat{F}(H) \circ \hat{F}(\pi_{m-1}) \circ \Phi)(x))_n
\]

\[
\Rightarrow m = n + 1 \land [y]_n = [x]_n.
\]

Here the last bi-implication is a consequence of the fact that $x = m$ $(\Phi^{-1} \circ \hat{F}(B) \circ \hat{F}(H) \circ \hat{F}(\pi_{m-1}) \circ \Phi)(x)$ above. It is immediate from this that for the closure $A^{*}$ of $A$ we have

\[
(m, [x]_m) A^{*} (n, [y]_n) \iff m \geq n \land [y]_n = [x]_n.
\]

We know by definition that for $\psi \in K \times O \to 2$ we have $\psi = \square \psi$ if all $(m, [x]_m) \in K$ and all $\alpha \in O$ we have

\[
\psi((m, [x]_m), \alpha) = \square(\psi)((m, [x]_m), \alpha).
\]

But by our characterization of $A^{*}$ we have that the right hand side again equals

\[
\forall (n, [y]_n) \in K. (m, [x]_m) A^{*} (n, [y]_n) \Rightarrow \psi((n, [y]_n), \alpha)
\]

\[
= \forall n \leq m, \psi((n, [x]_m), \alpha).
\]

From this it is immediate that $\psi = \square \psi$ holds iff we have that $\psi \in K \times O \to \text{her}_2$ and we are done.

\[\square\]

\textit{Proof of Proposition A.7.} We shall consider only three of the cases.

\textbf{Constant Non-empty Sets} Let $X$ be some fixed non-empty set. Let $F : \text{set} \to \text{set}$ be the constant functor mapping any set to $X$ and any function to the identity map $1_X$. We need to come up with a plain lift $\hat{F} : \text{CBU}_{\text{her}} \to \text{CBU}_{\text{her}}$ of $F$. We naturally choose $\hat{F}$ to be the constant functor mapping any object of $\text{CBU}_{\text{her}}$ to $X$ equipped with the discrete metric $d_1$ and any morphism of $\text{CBU}_{\text{her}}$ to the identity map $1_X$. This easily constitutes a locally non-expansive functor $\hat{F} : \text{CBU}_{\text{her}} \to \text{CBU}_{\text{her}}$ and obviously is a lift of $F$. For any $\varphi : (Y, d) \to (Z, e)$ whatsoever we get that for any $m > 0$ and any two $x, y \in Y$, $d(x, d) = (X, d_1)$ with $x = m$, $y$ we have $x = y$, in particular we have that $\hat{F}(\varphi)x = 1_X(x) = 1_Y(y) = \hat{F}(\varphi)(y)$. Hence $\hat{F}$ is non-shrinking. Finally note that since $(X, d_1)$ is bisected we have that $\hat{F}$ maps all objects to bisected objects, in particular those that were bisected already.

\textbf{Products} Let us consider the case of products: we shall work with binary products only but the construction generalizes to any finite product. Take two functors $F, G : \text{set} \to \text{set}$ and define $H : \text{set} \to \text{set}$ by mapping a set $X$ to the set $F(X) \times G(X)$ and a map $\varphi : X \to Y$ to $F(\varphi) \times G(\varphi) : F(X) \times G(X) \to F(Y) \times G(Y)$. Under the assumption that we have plain lifts $\hat{F}, \hat{G} : \text{CBU}_{\text{her}} \to \text{CBU}_{\text{her}}$ of $F$ and $G$, we have to build a plain lift $H$ of $H$.

For an object $(X, d) \in \text{CBU}_{\text{her}}$ we write $(Y, e) = \hat{F}(X, d)$ and $(Z, f) = \hat{G}(X, d)$ and assign

\[
\hat{H}(X, d) = (Y \times Z, e \times f),
\]

where the product metric $e \times f$ on $Y \times Z$ is defined by $(e \times f)((y_0, z_0), (y_1, z_1)) = \max(e(y_0, y_1), f(z_0, z_1))$ for any two $(y_0, z_0), (y_1, z_1) \in Y \times Z$. For a morphism $\varphi : (X_0, d_0) \to (X_1, d_1) \in \text{CBU}_{\text{her}}$ we write $\hat{F}(\varphi) = (Y_0, e_0) \to (Y_1, e_1)$ and $\hat{G}(\varphi) = (Z_0, f_0) \to (Z_1, f_1)$ and assign

\[
\hat{H}(\varphi) = \hat{F}(\varphi) \times \hat{G}(\varphi) : Y_0 \times Z_0 \to Y_1 \times Z_1.
\]
It is well known that this yields a well-defined and locally non-expansive functor $\hat{H} : \text{CBUlt}_{\text{loc}} \to \text{CBUlt}_{\text{loc}}$. For the action on objects, this is spelled out in Lemmas 1.24 and 1.28 of [23].

We now proceed to prove that the functor $\hat{H}$ is indeed a plain lift of $H$. First up is the property of being a lift, take an object $(X, d)$ of $\text{CBUlt}_{\text{loc}}$. We write $(Y, e) = \hat{F}(X, d)$ and $(Z, f) = \hat{G}(X, d)$ and get that

$$U(\hat{H}(X, d)) = U(Y \times Z, e \times f) = U(\hat{F}(X, d)) \times U(\hat{G}(X, d)) = F(U(X, d)) \times G(U(X, d)) = H(U(X, d)).$$

For a morphism $\varphi : (X_0, d_0) \to (X_1, d_1) \in \text{CBUlt}_{\text{loc}}$ we get the weirdly easy calculation $U(\hat{H}(\varphi)) = \hat{H}(\varphi) = \hat{F}(\varphi) \times \hat{G}(\varphi) = F(\varphi) \times G(\varphi) = H(\varphi)$ since the forgetful functor $H$ has no action on morphisms.

Next up is proof that $\hat{H}$ is non-shrinking. Take a morphism $\varphi : (X_0, d_0) \to (X_1, d_1) \in \text{CBUlt}_{\text{loc}}$, we write $\hat{F}(\varphi) = (Y_0, e_0) \to (Y_1, e_1)$ and $\hat{G}(\varphi) = (Z_0, f_0) \to (Z_1, f_1)$. Assume that for some $m > 0$ we have that

$$\forall x, y \in X_0. x =_m y \Rightarrow \varphi(x) = \varphi(y).$$

Now take $(y_0, z_0), (y_1, z_1) \in Y_0 \times Z_0$ and assume that we have $(y_0, z_0) =_m (y_1, z_1)$. But then $y_0 =_m y_1$ and $z_0 =_m z_1$ by the definition of the metric $e_0 \times f_0$. And so we have

$$\hat{H}(\varphi)(y_0, z_0) = \left(\hat{F}(\varphi) \times \hat{G}(\varphi)\right)(y_0, z_0) = \left(\hat{F}(\varphi)(y_0), \hat{G}(\varphi)(z_0)\right) = \left(\hat{F}(\varphi)(y_1), \hat{G}(\varphi)(z_1)\right) = \left(\hat{F}(\varphi)(y_1), \hat{G}(\varphi)(z_1)\right) = \hat{H}(\varphi)(y_1, z_1)$$

since both $\hat{F}$ and $\hat{G}$ were assumed non-shrinking. Finally we remark that $\hat{H}$ preserves the property of being bisected since that holds by assumption for $\hat{F}$ and $\hat{G}$ and because the product metric introduces no new distances.

**Finite, Partial Maps from a Constant Set** Now on to finite, partial maps from a constant set. Take a set $X$ and a functor $F : \text{Set} \to \text{Set}$, define $G : \text{Set} \to \text{Set}$ by mapping a set $Y$ to the set $X \to f_{\text{fin}}(F(Y))$ of partial maps with a finite domain. A map $\varphi : Y \to Z$ is mapped to $\lambda \psi : X \to f_{\text{fin}}(F(Y)) \cdot (F(\varphi) \circ \psi)$. Under the assumption that we have a plain lift $\hat{F} : \text{CBUlt}_{\text{loc}} \to \text{CBUlt}_{\text{loc}}$ of $F$, we have to build a plain lift $\hat{G}$ of $G$.

For an object $(Y, e) \in \text{CBUlt}_{\text{loc}}$ we write $(Z, e) = \hat{F}(Y, d)$ and assign

$$\hat{G}(Y, d) = (X \to f_{\text{fin}}(Z), e_{X \to f_{\text{fin}}})$$

where $e_{X \to f_{\text{fin}}}(\psi_0, \psi_1)$ is the maximum over $\psi_0, \psi_1 : X \to f_{\text{fin}}(Z)$ with identical domain, otherwise the distance is 1. For a morphism $\varphi : (Y_0, d_0) \to (Y_1, d_1) \in \text{CBUlt}_{\text{loc}}$ we write $\hat{F}(\varphi) : (Z_0, f_0) \to (Z_1, f_1)$ and employ that $\hat{F}$ is a lift of $F$ to simply assign

$$\hat{G}(\varphi) = G(\varphi) : (X \to f_{\text{fin}}(Z_0)) \to (X \to f_{\text{fin}}(Z_1)).$$

It is easily verifiable – if not exactly well known – that this yields a well-defined and locally non-expansive functor $\hat{G} : \text{CBUlt}_{\text{loc}} \to \text{CBUlt}_{\text{loc}}$, a high level argument is given in the proof of Proposition 22 of [18].

We now proceed to prove that the functor $\hat{G}$ is a plain lift of $G$. First we verify that it is a lift, take an object $(Y, d)$ of $\text{CBUlt}_{\text{loc}}$. We write $(Z, e) = \hat{F}(Y, d)$ and get that

$$U(\hat{G}(Y, d)) = U(X \to f_{\text{fin}}(Z), e_{X \to f_{\text{fin}}}) = X \to f_{\text{fin}}(Z)$$

$$= X \to f_{\text{fin}}(\hat{F}(Y, d)) = X \to f_{\text{fin}}(U(\hat{F}(Y, d))) = G(U(Y, d)).$$

The case of morphisms holds by definition.

Next up is proof that $\hat{G}$ is non-shrinking. Take a morphism $\varphi : (Y_0, d_0) \to (Y_1, d_1) \in \text{CBUlt}_{\text{loc}}$, we write $\hat{F}(\varphi) = (Z_0, f_0) \to (Z_1, f_1)$. Assume that for some $m > 0$ we have that

$$\forall x, y \in Y_0. x =_m y \Rightarrow \varphi(x) = \varphi(y).$$

Now take $\psi_0, \psi_1 \in X \to f_{\text{fin}}(Z_0)$ and assume that we have $\psi_0 =_m \psi_1$. We have $\text{dom}(\psi_0) = \text{dom}(\psi_1)$ and furthermore know that for all $x \in \text{dom}(\psi_0)$ we have $\psi_0(x) = \psi_1(x)$. We obviously have $\text{dom}(G(\varphi)(\psi_0)) = \text{dom}(\psi_0) = \text{dom}(\psi_1) = \text{dom}(G(\varphi)(\psi_0))$ and for any $x$ in this domain we get

$$G(\varphi)(\psi_0)(x) = G(\varphi)(\psi_1)(x) = G(\varphi)(\psi_1)(x)$$

Finally we remark that $\hat{G}$ preserves the property of being bisected since that holds by assumption for $\hat{F}$ and because we introduce no new distances by taking a maximum of finitely many existing distances. 

**C. Proofs about Capabilities**

The interpretation of types and capabilities satisfies standard substitution properties.

**Lemma C.1.** If $\eta$ and $\eta'$ agree on the free variables of $\tau$, $\theta$ and $C$ then $[\tau]_{\eta} = [\tau]_{\eta'}$, $[\theta]_{\eta} = [\theta]_{\eta'}$, and $[C]_{\eta} = [C]_{\eta'}$.

**Lemma C.2.** We have $[\tau[\alpha:=\tau]]_{\eta} = [\tau[\alpha:=\tau]]_{\eta'}$, and analogous substitution properties hold for capabilities and memory types.

**Lemma C.3.** For all $S \in M_T$ and $c \in \text{Cap}$,

$$(S \otimes \iota(c) * c)(w) * \iota^{-1}(w)(\text{emp}) = S(w') * \iota^{-1}(w')(\text{emp})$$

for $w' = \iota(c) * c$ or $w$. Moreover, $\mathcal{E}(S \otimes \iota(c) * c) = \mathcal{E}(S) \otimes \iota(c)$.

**Proof.** Let $S \in M_T$, $c \in \text{Cap}$, $w \in W$ and set $w' = \iota(c) * c$. By definition of $\iota$ we have

$$(w') * \iota^{-1}(w')(\text{emp}) = S(\iota(c) * c)(w) * \iota^{-1}(\iota(c) * c)(w)(\text{emp})$$

$$(S \otimes \iota(c) * c)(w) * \iota(c) * c(\text{emp}) * \iota^{-1}(w)(\text{emp})$$

$$(S \otimes \iota(c) * c)(w) * \iota(c) * c(\text{emp}) * \iota^{-1}(w')(\text{emp})$$

$$(S \otimes \iota(c) * c)(w) * \iota^{-1}(w')(\text{emp})$$

Since $w$ above was chosen arbitrarily, the second statement, $\mathcal{E}(S \otimes \iota(c) * c) = \mathcal{E}(S) \otimes \iota(c)$, is an immediate consequence by the definition of $\mathcal{E}$ and $\otimes$. 

$\square$
C.1 Structural Equivalence of Capabilities and Types

Capabilities and types are considered up to the structural equivalence given in Figure 6. In this subsection we check that our semantics of types respects this equivalence.

Note that the abbreviation defined in (4) gives a connective that is interpreted by the operation $\circ$ on $W$, up to the isomorphism between $W$ and $\text{Cap}$. More precisely, by definition of $\circ$ we have

$$\eta^{-1}(\iota(c_1) \circ \iota(c_2))(w) = \eta^{-1}(\iota(c_1))((\iota(c_2) \circ w) \ast \eta^{-1}(\iota(c_2))(w)) = (c_1 \ast (\iota(c_2))(w)) \ast c_2(w) = (c_1 \ast (\iota(c_2))(w)) \ast c_2(w)$$

for all $c_1, c_2 \in \text{Cap}$ and all $w$, and thus

$$\eta^{-1}(C_1 \circ C_2) = \eta^{-1}(C_1) \circ \eta^{-1}(C_2)$$

Hence, equations (5), (6) and (10) follow from Proposition 3.3. That (7)–(9) and (11) hold is a direct interpretation of the interpretation of $\ast$ in terms of the commutative monoid structure on $\text{Cap}$ and the associated monoid action (Proposition 3.4).

Next, we consider the action by $\ast$ on capabilities and contexts.

Lemma C.4 ($\ast$-distribution axiom for singleton capability). The following equivalence holds with respect to the semantics:

$$\{ \sigma : \emptyset \} \ast C = \{ \sigma : \emptyset \ast C \}$$

Proof. We calculate as follows.

$$\{ \sigma : \emptyset \} \ast C\eta w = \{ \{ \sigma \} : \emptyset \} w \ast [C]_\eta w = \{(k, h) | (k, (\eta \ast h)) \in [\emptyset] w \} \ast [C]_\eta w = \{(k, h \cdot h') | (k, (\eta \ast h)) \in [\emptyset \ast w] w \land (k, h') \in [C]_\eta w \} = \{ \sigma : \emptyset \ast C \} \eta w$$

Since this holds for arbitrary $\eta$ and $w$ we have proved (12). □

Lemma C.5 ($\ast$-distribution axiom for linear environments). The following equivalences hold with respect to the semantics:

$$(\Gamma, x : C) \ast C = \Gamma, x : (\emptyset \ast C)$$

(13)

$$(\Gamma, x : C) \ast C = (\Gamma \ast C), x : C$$

(14)

Proof. The equivalences (13) and (14) follow since all three environments have the same interpretation: $[(\Gamma, x : C) \ast C]_\eta w$, $[\Gamma, x : (\emptyset \ast C)]_\eta w$, and $[(\Gamma \ast C), x : C]_\eta w$ consist of all those pairs $(k, (\rho \ast h))$ where we have

$$\rho = \rho_1[x \mapsto v]$$

$$h = h_1 \cdot h_2 \cdot h_3$$

$$(k, (\rho_1, h_1)) \in [\Gamma]_\eta w$$

$$(k, (v, h_2)) \in [X]_\eta w$$

$$(k, h_3) \in [C]_\eta w$$

for some $\rho_1$, $v$ and $h_1, h_2, h_3$.

We next consider the equivalences that describe the action by $\otimes$. The first group are the general equivalences (15) and (16) that apply to several syntactic categories. The first equation says that $\otimes$ distributes over $\ast$. Equation (16) states that $\otimes$ does not affect existential quantification over region names.

Lemma C.6 (General $\otimes$-distribution axioms). The following equivalences hold with respect to the semantics:

$$(\ast \circ \otimes) C = (\ast \otimes C) \ast (\ast \otimes C)$$

(15)

$$(\exists \sigma : \emptyset \circ \otimes) C = \exists \sigma : (\ast \otimes C)$$

if $\sigma \notin \text{RegNames}(C)$

(16)

Proof. In each case, we prove that the left hand side and right hand side have the same denotation. Equation (15) follows from the pointwise definition of $\ast$ with respect to worlds; we show this for the case of memory types:

$$\{ \sigma : \emptyset \ast C \} \circ [C]_\eta w = \{ \sigma \} [C]_\eta w \ast (\{ \sigma : \emptyset \ast C \} \circ C) w = \{ \sigma \} \ast [C]_\eta w \ast [C \ast C]_\eta w$$

Equation (16) follows from the interpretation of existential quantification. We show this again for the case of memory types:

$$\{ \exists \sigma : \emptyset \ast C \} \circ [C]_\eta w = \{ \exists \sigma : \emptyset \ast C \} \circ [C]_\eta w$$

Here, the third equation uses the assumption $\sigma \notin \text{RegNames}(C)$, and thus that $[C]_\eta$ does not depend on the value of $\eta$ on $\sigma$. □

The next group of equivalences (17) and (18) describes the interaction of $\otimes$ with the remaining capabilities. Together with (10), (15), (16) and the unfolding of recursively defined capabilities, they cover all the cases.

Lemma C.7 ($\otimes$-distribution axiom for singleton capabilities). The following equivalences hold:

$$\emptyset \ast C = \emptyset$$

(17)

$$\{ \sigma : \emptyset \} \ast C = \{ \sigma : \emptyset \ast C \}$$

(18)

Proof. Equivalence (17) follows from the interpretation of $\emptyset$ as a constant function:

$$\emptyset \ast C\eta w = \{ \emptyset \} \ast (\{ C \} \circ w) = \emptyset \times \text{Heap} = \emptyset\eta w$$

For (18) we note that

$$\{ \sigma : \emptyset \} \ast C\eta w = \{ \{ \sigma : \emptyset \} \ast (\{ C \} \circ w) = \{(k, h) | (k, (\sigma \ast h)) \in [\emptyset \ast w] \ast (\{ C \} \circ w) = \{(k, h) | (k, (\sigma \ast h)) \in [\emptyset \ast C]_\eta w$$

Since this holds for arbitrary $w$, the equivalence of the left hand side and right hand side of (18) follows. □

Next, we consider the equivalences that describe the interaction between $\otimes$ and the value type constructors.

17

2010/4/6
Lemma C.8 ($\otimes$-distribution axioms for value types). The following equivalences hold:

\begin{align*}
0 \otimes C &= 0 \\
1 \otimes C &= 1 \\
\text{int} \otimes C &= \text{int} \\
(\tau_1 + \tau_2) \otimes C &= (\tau_1 \otimes C) + (\tau_2 \otimes C) \\
(\tau_1 \times \tau_2) \otimes C &= (\tau_1 \otimes C) \times (\tau_2 \otimes C) \\
(\forall \xi \cdot r) \otimes C &= \xi \not\in fvc \quad (\forall \xi \cdot r \otimes C) \\
(\chi_1 \rightarrow \chi_2) \otimes C &= (\chi_1 \otimes C) \rightarrow (\chi_2 \otimes C) \\
[\sigma] \otimes C &= [\sigma] 
\end{align*}

Proof. Equations (19), (20), (21) and (26) follow since universal quantification over capabilities: We show that $r \chi \text{Set} \tau J$ and let $J C w$ are all constant functions. The equation (22) for sum types uses the pointwise definition of $[\tau_1 + \tau_2]$:

$$
[(\tau_1 + \tau_2) \otimes C]_\eta w = [(\tau_1 + \tau_2)]_\eta (\iota([C]_\eta) \circ w)
$$

$$
= \{(k, \text{inj}_1' v) \mid (k, v) \in [\tau_1]_\eta, (\iota([C]_\eta) \circ w)\}
$$

$$
= \{(k, \text{inj}_2' v) \mid (k, v) \in [\tau_1 \otimes C]_\eta w\}
$$

$$
= ([\tau_1 \otimes C]_\eta + (\tau_2 \otimes C)]_\eta w
$$

Equation (23) is proved similarly. For (24) we consider the case of universal quantification over capabilities:

$$
[\forall \gamma \cdot r] \otimes C]_\eta w = [\forall \gamma \cdot r \otimes C]_\eta (\iota([C]_\eta) \circ w)
$$

$$
= \bigcap_{c \in \text{Cap}} [\forall \gamma \cdot r]_\eta (\iota([C]_\eta) \circ w)
$$

$$
= \bigcap_{c \in \text{Cap}} [\forall \gamma \cdot r]_\eta (\iota([C]_{\gamma = c}) \circ w)
$$

$$
= \bigcap_{c \in \text{Cap}} [\forall \gamma \cdot C]_\eta w
$$

$$
= [\forall \gamma \cdot (r \otimes C)]_\eta w
$$

Here, the third equality is by Lemma C.1 and assumption $\gamma \not\in fvc C$.

The most interesting equivalence is the distribution axiom for function types, equation (25). However, the semantics of function types is set up such that this axiom can be proved fairly straightforwardly: first note that

$$
[\chi_1 \rightarrow \chi_2] \otimes C]_\eta w = [\chi_1 \rightarrow \chi_2]_\eta (\iota([C]_\eta) \circ w)
$$

Assume $(k, v) \in [\chi_1 \rightarrow \chi_2] \otimes C]_\eta w$. Thus, for all $j < k$, and for all $r' \in \text{Cap}$: if $w'' = (\iota([C]_\eta) \circ w)$ then

$$
\forall (j, (v', h)) \in [\chi_1]_\eta w'', r'(w'') \ast \eta^{-1}(w'(\text{emp})).
$$

$$(j + 1, (v', h)) \in \mathcal{E}([\chi_2]_\eta r')(w').
$$

We show that $(k, v) \in [\chi_1 \otimes C] \rightarrow (\chi_2 \otimes C)]_\eta w$. So let $j < k$, $r \in \text{Cap}$ and let

$$
(j, (v', h)) \in [\chi_1 \otimes C]_\eta w \ast r(w') \ast \eta^{-1}(w'(\text{emp})�)
$$

Since $\chi_1 \otimes C$ abbreviates $(\chi_1 \otimes C) \ast C$ and

$$
[\chi_1 \otimes C] \ast C]_\eta w \ast \eta^{-1}(w'(\text{emp})) = \mathcal{E}([\chi_1]_\eta w') \ast \eta^{-1}(w'(\text{emp}))
$$

by Lemma C.3, (51) is equivalent to

$$
(j, (v', h)) \in [\chi_1]_\eta w' \ast r(w') \ast \eta^{-1}(w'(\text{emp})�).
$$

Set $r'(w_0) = r(w)$, then (50) yields

$$(j + 1, (v', h)) \in \mathcal{E}([\chi_2]_\eta r')(w').
$$

and with Lemma C.3 we can use

$$
[\chi_1]_\eta w' \ast \eta^{-1}(w'(\text{emp})�) \ast r'(w') = [\chi_2 \otimes C]_\eta w \ast \eta^{-1}(w')(\text{emp}) \ast r(w)
$$

to conclude $\mathcal{E}([\chi_2]_\eta r')(w') = \mathcal{E}([\chi_2 \otimes C]_\eta r)(w)$. Hence we have shown $(k, v) \in [\chi_1 \otimes C] \rightarrow (\chi_2 \otimes C)]_\eta w$. The other direction is analogous.

Lemma C.9 ($\otimes$-distribution axioms for memory types). The following equivalences hold:

\begin{align*}
(\theta_1 + \theta_2) \otimes C &= (\theta_1 \otimes C) + (\theta_1 \otimes C) \\
(\theta_1 \times \theta_2) \otimes C &= (\theta_1 \otimes C) \times (\theta_2 \otimes C) \\
(\text{ref } \theta) \otimes C &= \text{ref } (\theta \otimes C)
\end{align*}

Moreover, the inclusion of value types into memory types commutes with $\cdot \otimes C$.

Proof. Equivalence (27) is an easy verification:

$$
[(\text{ref } \theta) \otimes C]_\eta w = [\text{ref } \theta]_\eta (\iota([C]_\eta) \circ w)
$$

$$
= \{(k, (l \cdot v, h)) \mid (k, (v, h)) \in [\theta]_\eta (\iota([C]_\eta) \circ w)\}
$$

$$
= \{(k, (l \cdot v, h)) \mid (k, (v, h)) \in [\theta \otimes C]_\eta w\}
$$

Equivallences (22) and (23) are similar. Finally,

$$
[(\tau \otimes C)]_\eta w = [\tau]_\eta (\iota([C]_\eta) \circ w)
$$

$$
= \{(k, (v, h)) \mid h \in \text{Heap}, (k, v) \in [\tau]_\eta (\iota([C]_\eta) \circ w)\}
$$

$$
= \{(k, (v, h)) \mid h \in \text{Heap}, (k, v) \in [\tau \otimes C]_\eta w\}
$$

shows that invariant extension commutes with the inclusion of value types $\tau$ into memory types.

Lemma C.10 ($\otimes$-distribution axioms for environments). The following equivalences hold:

\begin{align*}
\otimes C &= \otimes C \\
(\Gamma, x: \chi) \otimes C &= (\Gamma \otimes C), x: (\chi \otimes C) \\
(\Gamma \ast C) \otimes C &= (\Gamma \otimes C) \ast (C' \otimes C)
\end{align*}

Proof. Equation (28) follows since $\otimes$ is a constant function; equations (29) and (30) hold due to the pointwise interpretation.

Lemma C.11 (Distribution axioms for region abstraction). The following equivalences hold:

\begin{align*}
\exists \sigma_1, \exists \sigma_2, \cdot &= \exists \sigma_2, \exists \sigma_1, \cdot \\
\cdot \ast (\exists \sigma.C) &= \exists \sigma, \cdot \ast C \\
\{ \sigma_1 : \exists \sigma_2, \theta \} &= \exists \sigma_2, \{ \sigma_1 : \theta \}
\end{align*}

where $\sigma_1 \neq \sigma_2$

Proof. Equation (31) follows from the semantics of existential quantification; we show this for the case of a capability $C$:

$$
[\exists \sigma_1 \cdot \exists \sigma_2.C]_\eta w = \bigcup [C]_{\eta_{\sigma_1 \cdot \sigma_2} = v_1 \cdot v_2} w \mid v_1, v_2 \in \text{Val}
$$

$$
= [\exists \sigma_2, \exists \sigma_1.C]_\eta w
$$
Similarly, we verify (32) for a capability \( C_1 \) such that \( \sigma \not\in \text{RegNames}(C_1) \):

\[
[C_1 \ast \langle \exists \sigma \rangle C \rangle]_w = \bigcup \{ [C_1]_v \ast \{ [C]_v \mid v \in \text{Val} \} \}
\]

Finally, we consider (33):

\[
[[\sigma : \exists \sigma_2 \theta]]_w = \{ (k, h) \mid (k, (\eta \sigma_1, h)) \in [[\exists \sigma_2 \theta]]_w \}
\]

where the third step holds since we assumed \( \sigma_1 \neq \sigma_2 \).

\begin{lemma}
(Axioms for focusing) The following equivalences hold:

\[
\{ \sigma_1 : \text{ref } \theta \} = \exists \sigma_2, \{ \sigma_1 : \text{ref } [\sigma_2] \} \ast \{ \sigma_2 : \theta \}
\] (34)

\[
\{ \sigma : \theta_1 \ast \theta_2 \} = \exists \sigma_1, \{ \sigma : \theta_1 \} \ast \{ \sigma_1 : \theta_2 \}
\] (35)

\[
\{ \sigma : \theta_1 \ast \theta_2 \} = \exists \sigma_2, \{ \sigma : \theta_1 \} \ast \{ \sigma_2 : \theta_2 \}
\] (36)

\[
\{ \sigma : \theta_1 \ast \theta_2 \} = \exists \sigma_1, \{ \sigma : \theta_1 \ast \theta_2 \}
\] (37)

\[
\{ \sigma : \theta_1 \ast \theta_2 \} = \exists \sigma_2, \{ \sigma : \theta_1 \ast \theta_2 \}
\] (38)

\end{lemma}

\begin{proof}
For equation (34) we can calculate as follows.

\[
\bigcup \{ [\sigma_1 : \text{ref } [\sigma_2]]_v \ast \{ [\sigma_2 : \theta]_v \} \mid v \in \text{Val} \}
\]

For equation (35) we calculate similarly:

\[
[\exists \sigma_1, \{ \sigma : \theta \} \ast \{ \sigma_1 : \theta \}]_w
\]

Finally, the axioms (39)–(41) hold as a consequence of the substitution lemma C.2 and our use of equivocative types.

\begin{lemma}
(Subtyping) This section gives the proofs of the subtyping axioms in Figure 7.

\end{lemma}

\begin{proof}

\end{proof}
Proof. We must show that for any \( w \) and \( \eta \), if \([(k, (v, h))] \in [\tau]_w \) then \([(k, (v, h))] \in [\sigma \cdot \tau]_w \). Let \( \eta' = \eta[\sigma = v] \). By definition of the existential quantification over region names, it suffices to show that \([(k, (v, h))] \in [\sigma \cdot \tau]_w \). Since we have \( \eta'(\sigma) = v \) it is clear that \([(k, v)] \in [\sigma]_w \). The remaining proof obligation \([(k, h)] \in [\sigma \cdot \tau]_w \) follows since \([\tau]_w = [\tau]_{\eta'} \), due to the assumption that \( \sigma \) is not free in \( \tau \). 

Lemma C.16 (Singleton extraction). The following subtyping axiom holds:

\[
[\sigma] \cdot [\sigma \cdot \tau] \subseteq \tau \cdot [\sigma \cdot \tau]
\]

Proof. We must show that for any \( w \) and \( \eta \), if \( [(\sigma \cdot \tau)]_w \) is contained in \( [\tau \cdot [\sigma \cdot \tau]]_w \). So assume \([(k, v)] \in [\sigma]_w \) is in the former set. By definition, this means \( (k, v) \in [(\sigma \cdot \tau)]_w \) and \([(k, h)] \in [\sigma \cdot \tau]_w \). The first property gives \( \eta' \cdot \sigma = v \). But then the second property yields \( [(k, v)] \in [\tau]_w \), and thus \( (k, v) \in [\tau]_w \). Combining these facts, we get \( (k, (v, h)) \in [\tau \cdot [\sigma \cdot \tau]]_w \).

C.3 Value and expression typing judgements

Most of the typing rules for values are verified straightforwardly with respect to the step-indexed semantics.

Lemma C.17 (VAR). Suppose \( (x : \tau) \in \Delta \). Then, for any \( \eta \), \( \eta \vdash (\Delta \vdash x : \tau) \).

Proof. Let \( w \in W \), \( k \in \mathbb{N} \) and \( (k, \rho) \in [\Delta]_w \). Since \( x : \tau \in \Delta \) we immediately obtain the required \( (k, \rho(x)) \in [\tau]_w \) from the definition of \([\Delta]\).

Lemma C.18 (UNIT). For any \( \eta \), \( \eta \vdash (\Delta \vdash () : 1) \).

Proof. Let \( w \in W \), \( k \in \mathbb{N} \) and \( (k, \rho) \in [\Delta]_w \). We must show that \( (k, \rho()) \in [1]_w \), which is immediate by the definition of \([1]\).

Lemma C.19 (INJ). Suppose \( \eta \vdash (\Delta \vdash v : \tau_1) \). Then \( \eta \vdash (\text{inj}^i v : \tau_1 + \tau_2) \).

Proof. Let \( i = 1 \) or \( i = 2 \). Let \( w \in W \), \( k \in \mathbb{N} \) and \( (k, \rho) \in [\Delta]_w \). We must show that \( (k, \rho(\text{inj}^i v)) \in [\tau_1 + \tau_2]_w \). Unfolding the computation, we have \( (k, v) \in [\tau_1]_w \), and thus \( (k-1, \rho(v)) \in [\tau_1]_w \) by the uniformity of \([\tau_1]_w \). Using \( \rho(\text{inj}^i v) = \text{inj}^i \rho(v) \) and the definition of \([\tau_1 + \tau_2]_w \), we obtain \( (k, \rho(\text{inj}^i v)) \in [\tau_1 + \tau_2]_w \). 

Lemma C.20 (PAIR). Suppose \( \eta \vdash (\Delta \vdash v_1 : \tau_1) \) and \( \eta \vdash (\Delta \vdash v_2 : \tau_2) \). Then \( \eta \vdash (\Delta \vdash (v_1, v_2) : \tau_1 \times \tau_2) \).

Proof. Let \( w \in W \), \( k \in \mathbb{N} \) and \( (k, \rho) \in [\Delta]_w \). We must show \( (k, v_1), (v_2) \in [\tau_1 \times \tau_2]_w \), i.e., \( (k, \rho(v_1)), (k, \rho(v_2)) \in [\tau_1 \times \tau_2]_w \). By assumption, we have both \( (k, \rho(v_1)) \in [\tau_1]_w \) and \( (k, \rho(v_2)) \in [\tau_2]_w \), which by the uniformity of \([\tau_1]_w \) and \([\tau_2]_w \) suffices.

Lemma C.21 (RECFUN). Suppose \( \eta \vdash (\Delta, f : \chi_1 \rightarrow \chi_2, x : \chi_1 \vdash f : \chi_2) \). Then \( \eta \vdash (\Delta \vdash \mu f. \lambda x.t : \chi_1 \rightarrow \chi_2) \).

Proof. Let \( w \in W \). We prove by induction on \( k \):

\[
\forall k \in \mathbb{N}. \ (k, \rho) \in [\Delta]_w \Rightarrow (k, \rho(\mu f. \lambda x.t)) \in [\chi_1 \rightarrow \chi_2]_w
\]

In the case \( k = 0 \) there is nothing to show, since \( (0, v) \in [\chi_1 \rightarrow \chi_2]_w \) holds for any value \( v \) by definition of \([\chi_1 \rightarrow \chi_2]\). So suppose \( k > 0 \), let \( (k, \rho) \in [\Delta]_w \), let \( j < k \), let \( r \in \text{Cap} \) and \( (j, (v, h)) \in [\chi_1]_r + r(w) \). Then we can conclude

\[
(j+1, ((\mu f. \lambda x. t) v), h)) \in [\chi_2]_r + r(w). \quad (52)
\]

By the operational semantics:

\[
((\mu f. \lambda x. t) v) h \mapsto ((\mu f = \mu f. \lambda x. t, x = v)(t) h) \quad (53)
\]

By uniformity of \([\Delta]_w \), the induction hypothesis and uniformity of \([\chi_1 \rightarrow \chi_2]_w \) we have

\[
(j, \mu f. \lambda x. t) v \in [\chi_1 \rightarrow \chi_2]_w
\]

Thus, by the uniformity of \([\chi_1]_w \), we obtain

\[
(j, ((\mu f = \mu f. \lambda x. t, x = v)(t) h)) \in [\chi_1 \rightarrow \chi_2]_w
\]

Hence we have

\[
(j, ((\mu f = \mu f. \lambda x. t, x = v)(t) h)) \in [\chi_2]_r + r(w) \quad (54)
\]

by the assumption \( \eta \vdash (\Delta, f : \chi_1 \rightarrow \chi_2, x : \chi_1 \vdash f : \chi_2) \). From (54) and (53) we can conclude (52).

Lemma C.22 (\( \gamma\)-INTRO). Suppose \( \eta \vdash (\Delta \vdash v : \tau) \) and \( \alpha \) not free in \( \Delta \). Then \( \eta \vdash (\Delta \vdash v : \forall \alpha. \tau) \).

Proof. Suppose \( \eta \vdash (\Delta \vdash v : \forall \alpha. \tau) \). Then, for any \( \tau' \), we have that \( \eta \vdash (\Delta \vdash v : \forall \alpha. \tau' \rightarrow \tau) \).

Lemma C.24 (VAL). Suppose \( \eta \vdash (\Delta \vdash v : \tau) \).

Proof. Let \( w \in W \), \( k \in \mathbb{N} \), \( r \in \text{Cap} \) and \( (k, (\rho, h)) \in [\Delta]_w \). Using \( \rho(v) \) and the definition of \([\tau]_w \), we obtain \( (k, \rho(v)) \in [\tau]_w \). 

Hence we have

\[
(\rho(v), h) \in [\tau]_w \quad (55)
\]

by the assumption \( \eta \vdash (\Delta, f : \chi_1 \rightarrow \chi_2) \) and \( \eta \vdash (\Delta, \Gamma \vdash t : \chi_1) \). Then \( \eta \vdash (\Delta, \Gamma \vdash v : \chi_2) \).

Proof. Let \( w \in W \), \( k \in \mathbb{N} \), \( r \in \text{Cap} \). Let \( \eta \vdash (\Delta \vdash v : \forall \alpha. \tau) \) and let \( \rho_1 \) be the restriction of \( \rho \) to dom \( \Delta \). Thus, \( (k, \rho_1) \) \( \in [\Delta]_w \). By the assumptions and using \( \rho(v) = \rho_1 v \) we obtain

\[
(k, \rho_1 v) \in [\chi_1 \rightarrow \chi_2]_w \quad (56)
\]

We must show

\[
(\rho(v), h) \in [\chi_1 \rightarrow \chi_2]_w \quad (57)
\]

To this end, assume that \( ((\rho(v), h) \mapsto (t' h')) \) for some \( j \leq k \) and irreducible \( (t' h') \). By the determinacy of the operational semantics (up to the choice of location names), there exists \( i \leq j \) such that this sequence decomposes into

\[
((\rho(v), h) \mapsto (t' h')) \mapsto \cdots \mapsto (t'' h'') \mapsto \cdots \mapsto (t' h')
\]

2010/4/6
where \((\rho \cdot h) \mapsto t^1\ \text{and} \ (t^2 \mid h^2)\) is irreducible. By (56) and the uniformity of \([x_1 \eta]_w\) we know
\[
(k \prec i - 1, (t''', h''') \prec \Sigma \cdot [x_1]_w w \ast r(w) \ast r^{-1}(w)(\text{emp}) \cdot \]

By (55) this yields \((k \prec i, (p''', h''') \prec \Sigma)\) \(\in \Sigma\ [x_2]_w r(w)\), and from \(j \prec i \leq k - i \prec i\) we therefore have
\[
(k \prec i - (j \prec i), (t', h') \prec \Sigma) \in [x_2]_w w \ast r(w) \ast r^{-1}(w)(\text{emp}) \cdot \]

In summary, we have shown (57). □

**Lemma C.26 (SHALLOW-FRAME).** Suppose \(\eta \models (\Gamma \models \top : \chi)\). Then \(\eta \models (\Gamma \models C \models \top : \chi \cap C)\).

**Proof.** Let \(w \in W, k \in \mathbb{N}\) and \(r \in \text{Cap}\). Let
\[
(k, (\rho, h)) \in [\Gamma \models C}_w w \ast r(w) \ast r^{-1}(w)(\text{emp}) \cdot \]

This can be written equivalently as
\[
(k, (\rho, h)) \in [\Gamma \models C}_w w \ast r(w) \ast r^{-1}(w)(\text{emp}) \cdot \]

for \(r' = [\Gamma \models C}_w r\). The assumption \(\eta \models (\Gamma \models \top : \chi)\) then yields
\[
(k, (\rho, h)) \in [\Gamma \models C}_w r(w) \cdot \]

and thus, by unfolding \(r'\),
\[
(k, (\rho, h)) \in E([\chi \cap C]_w r(w) \cdot \)

In summary, we have shown \(\eta \models (\Gamma \models C \models \top : \chi \cap C)\). □

**Lemma C.27 (DEEP-FRAME).** Suppose \(\eta \models (\Gamma \models \top : \chi)\). Then \(\eta \models (\Gamma \models C \models \top : \chi (\cap C \models C)\).

**Proof.** A proof of this inference rule is given in an abstract setting in [37, Sect. 3]. Here we give a direct proof: Let \(w \in W, k \in \mathbb{N}\) and \(r \in \text{Cap}\). Let
\[
(k, (\rho, h)) \in [\Gamma \models C}_w w \ast r(w) \ast r^{-1}(w)(\text{emp}) \cdot \]

Note that this can be written equivalently as
\[
(k, (\rho, h)) \in [\Gamma \models C}_w w \ast r(w) \ast r^{-1}(w)(\text{emp}) \cdot \]

for \(r' = [\Gamma \models C}_w r\). By Lemma C.3. Then the assumption \(\eta \models (\Gamma \models \top : \chi)\) yields
\[
(k, (\rho, h)) \in E([\chi \cap C]_w r(w) \cdot \)

By the definition of \(r'\), \(r'\) and Lemma C.3,
\[
(k, (\rho, h)) \in E([\chi \cap C]_w r' r([\Gamma \models C]_w w) \cdot \]

\[
E(\Sigma, [\gamma \cap C]_w r' \cdot \Gamma \models C) \cdot \]

We have shown \(\eta \models (\Gamma \models C \models \top : \chi (\cap C \models C)\).

**Lemma C.28 (SUB).** Suppose \(\eta \models (\Gamma \models \top : \chi_1)\) and \(\chi_1 \subseteq \chi_2\). Then \(\eta \models (\Gamma \models \top : \chi_2)\).

**Proof.** As shown in Section C.2, \(\chi_1 \subseteq \chi_2\) implies \([x_1]_w w \subseteq [x_2]_w w\) for all \(\eta\) and \(w\). This gives us the soundness of SUB. □

**Lemma C.29 (PROJ-1).** Suppose \(\eta \models (\Gamma \models \top : \Sigma) \cdot \Sigma : \chi_1 \times \chi_2\).

Then \(\eta \models (\Gamma \models \top : \Sigma \ast \chi_1 \times \chi_2)\).

**Proof.** Let \(k \in \mathbb{N}\) and \(w \in W\). Let \(r \in \text{Cap}\) and let
\[
(k, (\rho, h)) \in [\Gamma \models C}_w w \ast r(w) \ast r^{-1}(w)(\text{emp}) \cdot \]

By assumption,
\[
(k, (\rho, h)) \in E([[\eta]_w, \Sigma : \chi_1 \times \chi_2]) \cdot \Sigma : \chi_1 \times \chi_2\]

Since \(\rho(v)\) is a value and therefore \((\rho(v) \cdot h)\) is irreducible, this means by definition that \(h\) splits into \(h = h' \cdot h''\) and that
\[
\eta \models (\rho(v)) \cdot h' \cdot h''\]

Thus there exist \(v_1, v_2 \in \mathcal{V}\) such that \(\rho(v) = (v_1, v_2)\) and
\[
(k-\chi_1, (v_1, v_2)) \in [\theta_1]_w w\]

By definition of the operational semantics, \(\text{proj}_1((\rho(v) \cdot h)) \mapsto (v_1 \mid h)\), and by the above considerations, \((k-\chi_1, (v_1, h' \cdot h''))\) is in
\[
[\tau_1 \cdot \{\sigma : \chi_1 \times \chi_2\}]_w w \ast r(w) \ast r^{-1}(w)(\text{emp}) \cdot \]

Thus, \((k, (\text{proj}_1(v'), h')) \in [\tau_1 \cdot \{\sigma : \chi_1 \times \chi_2\}]_w w \ast r(w) \ast r^{-1}(w)(\text{emp}) \cdot \]

The case of PROJ-2 is analogous to Lemma C.29.

**Lemma C.30 (CASE).** Suppose \(\eta \models (\Delta \models \top : \{\exists \chi_1, [\sigma] \models \{\sigma : [\sigma_1] = 0 \ast \{\sigma_1 : \chi_1 \models C) \rightarrow \chi_2\}}) \cdot \)

Since the restriction \(\rho_1\) of \(\rho\) to the domain of \(\Delta\) satisfies \((k, \rho_1) \in [\Delta]_w w\), the assumptions give the following properties:
\[
(k, \rho(v_1)) \in ([\exists \chi_1, [\sigma] \models \{\sigma : [\sigma_1] = 0 \ast \{\sigma_1 : \chi_1 \models C) \rightarrow \chi_2\})_w w\]

Unfolding the definitions, we obtain a splitting \(h = h' \cdot h''\) such that
\[
(k, (\eta, h')) \in [\theta_1 + \theta_2]_w w \ast r(w) \ast r^{-1}(w)(\text{emp}) \cdot \]

and thus \(\rho(v) = \eta(\sigma) = \text{inj}(v''')\) for some \(i = 1\) or \(i = 2\) and some \(v''\) such that
\[
(k-\chi_1, (v''', h'')) \in [\theta_1]_w w\]

Let us assume \(i = 1\); the other case is analogous. Further, let us write \(\eta_1\) for \(\eta_1 : v''\). Then, assuming \(\sigma_1 \not\in \text{RegNames}(C)\) and using uniformity, we have
\[
(k-\chi_1, (v''', h'')) \in [\theta_1]_w w \ast r(w) \ast r^{-1}(w)(\text{emp}) \cdot \]

Thus, we have \((k-\chi_1, (v'', h'')) \in [\exists \chi_1, [\sigma] \models \{\sigma : [\sigma_1] = 0 \ast \{\sigma_1 : \chi_1 \models C) \rightarrow \chi_2\}] w \ast r(w) \ast r^{-1}(w)(\text{emp}) \cdot \]

2010/46
By the above considerations for \( v_1 \) we obtain
\[
( k - 1, (\rho(\text{case}(v_1, v_2, v)), h) ) \in \mathcal{E}(\eta, r)(w).
\]
Since \((\rho(\text{case}(v_1, v_2, v)), h) \mapsto (\rho(v_1)') \mid h)\) by the operational semantics, this gives
\[
( k, (\rho(\text{case}(v_1, v_2, v)), h) ) \in \mathcal{E}(\eta, r)(w).
\]
In summary, we have shown \( \eta \vdash (\Delta, \Gamma \vdash \text{case}(v_1, v_2, v) : \chi) \).

**Lemma C.31** (Ref). Suppose \( \eta \vdash (\Gamma \vdash v : \tau) \).

Then \( \eta \vdash (\Gamma \vdash \text{ref} : \sigma \equiv \{ \sigma : \text{ref} \tau \}) \).

**Proof.** Let \( k \in \mathbb{N} \) and \( w \in W \), \( r \in \text{Cap} \) and
\[
( k, (\rho, h) ) \in \mathcal{E}(\eta, w \cdot r(w) \cdot \iota^{-1}(w)(\text{emp})).
\]
Then, \( \eta \vdash (\Gamma \vdash \text{ref} : \sigma \equiv \{ \sigma : \text{ref} \tau \}) \).

Thus, \( (k, \rho, v) \in \mathcal{E}(\tau, w) \) and \( (k, h, r) \in \mathcal{E}(w \cdot r(w) \cdot \iota^{-1}(w)(\text{emp})) \).

By uniformity, also \( (k - 2, \rho, v) \in \mathcal{E}(\tau, w) \).

By the assumption, we have
\[
( \text{ref}(\rho(v) \mid h) ) \mapsto (l \mid h \cdot ([l \mapsto (\rho(v)]) \mid h)
\]
for \( l \notin \text{dom} \ h \).

If we write \( \eta_1 \) for \( \eta_2 \) := \( l \) then we have
\[
( k - 1, l ) \in \mathcal{E}(\sigma, \eta_1)
\]
and thus also \( (k - 1, l \cdot [l \mapsto (\rho(v)]) ) \in \mathcal{E}(\sigma, \eta_1) \).

In summary, we have shown that \( (k - 1, l \cdot [l \mapsto (\rho(v)]) ) \in \mathcal{E}(\sigma, \eta_1) \).

Moreover, \( \eta \vdash (\Gamma \vdash \text{ref} : \sigma \equiv \{ \sigma : \text{ref} \tau \}) \).

**Lemma C.32** (GET). Suppose \( \eta \vdash (\Gamma \vdash \text{ref} \equiv \{ \sigma : \text{ref} \tau \}) \).

Then \( \eta \vdash (\Gamma \vdash \text{get} : \tau \equiv \{ \sigma : \text{ref} \tau \}) \).

**Proof.** Let \( k \in \mathbb{N} \) and \( w \in W \). Let \( r \in \text{Cap} \) and
\[
( k, (\rho, h) ) \in \mathcal{E}(\eta, w \cdot r(w) \cdot \iota^{-1}(w)(\text{emp})).
\]
By the assumption, \( \eta \vdash (\Gamma \vdash v : \sigma \equiv \{ \sigma : \text{ref} \tau \}) \), this means
\[
( k, (\rho, v, h) ) \in \mathcal{E}(\sigma, \text{ref} \tau, \eta, w \cdot r(w) \cdot \iota^{-1}(w)(\text{emp})).
\]
and therefore
\[
( k, (\rho, v, h) ) \in \mathcal{E}(\sigma, \text{ref} \tau, \eta, w \cdot r(w) \cdot \iota^{-1}(w)(\text{emp})).
\]
Then \( \eta \vdash (\Gamma \vdash \text{ref} \equiv \{ \sigma : \text{ref} \tau \}) \).

**Lemma C.33** (SET). Suppose \( \eta \vdash (\Gamma \vdash \text{set} \equiv \{ \sigma : \text{ref} \tau \}) \).

Then \( \eta \vdash (\Gamma \vdash \text{set} : \tau \equiv \{ \sigma : \text{ref} \tau \}) \).

**Proof.** Let \( k \in \mathbb{N} \) and \( w \in W \). Let \( r \in \text{Cap} \) and
\[
( k, (\rho, h) ) \in \mathcal{E}(\eta, w \cdot r(w) \cdot \iota^{-1}(w)(\text{emp})).
\]
By the assumption, \( \eta \vdash (\Gamma \vdash v : \sigma \equiv \{ \sigma : \text{ref} \tau \}) \), this gives
\[
( k, (\rho, v, h) ) \in \mathcal{E}(\sigma, \text{ref} \tau, \eta, w \cdot r(w) \cdot \iota^{-1}(w)(\text{emp})).
\]
Thus \( \rho(v) \) is of the form \( v = (v_1, v_2) \), \( \eta(\sigma) = \eta_1 \) and \( h' \) is of the form \( v_1 \mapsto v_1' \cdot h_0 \), for some values \( v_1, v_2 \) with \( (k - 1, v_2) \in \mathcal{E}(\tau, w) \).

In particular, \( (k - 1, v_1 \mapsto v_1' \cdot h_0) \in \mathcal{E}(\sigma, \text{ref} \tau, \eta, w) \).

By uniformity of \( \mathcal{E}(\tau, w) \) and therefore
\[
( k - 1, (v_1 \mapsto v_1' \cdot h'') ) \in \mathcal{E}(\sigma, \text{ref} \tau, \eta, r(w) \cdot \iota^{-1}(w)(\text{emp})).
\]
Since \( (k - 1, v_1 \mapsto v_1' \cdot h_0') \) is of the form \( v_1 \mapsto v_2 \cdot h_0 \cdot h'' \), this immediately yields
\[
( k, (\text{set}, v, h) ) \in \mathcal{E}(\sigma, \text{ref} \tau, \eta, r(w) \cdot \iota^{-1}(w)(\text{emp})).
\]
and we have shown \( \eta \vdash (\Gamma \vdash \text{set} : \tau \equiv \{ \sigma : \text{ref} \tau \}) \).
Expressions:
\[ e ::= x \mid 'C' \mid n \mid e_1 + e_2 \mid \ldots \quad (n \in \mathbb{Z}) \]

Commands:
\[ C ::= [e] \mid x \mapsto e \mid \{ x = [e] \}; C \mid \text{eval}[e] \mid \{ x = \text{new}(e) \}; C \mid \text{free } e \mid \text{skip} \mid C_1 ; C_2 \mid \text{if } (e_1 = e_2) \text{ then } C_1 \text{ else } C_2 \]

Figure 10. Programming language.

A quoted command, written ‘C’, and a new command for evaluating stored commands, written eval[e]. Informally, if the value of e is an integer n, and if the current heap contains the quoted command ‘C’ at location n, then the command eval[e] executes C as a subroutine.

We write fv(C) for the set of free variables of the command C, and similarly for expressions. Let V be the set of closed values of the language, and let H be the set of heaps, i.e., finite maps from integers to closed values:
\[ V = \mathbb{Z} \cup \{ C' : \text{fv}(C) = \emptyset \}, \]
\[ H = \mathbb{Z} \rightarrow \text{fin } V. \]

For two heaps h₁, h₂ ∈ H we write h₁ ∉ h₂ if they have disjoint domains and h₁ ⋈ h₂ for their union if this is the case. An environment is a finite map η from variables to closed values. When C is a command satisfying that fv(C) ⊆ η, we let η(C) denote the result of applying η to C as a capture-avoiding substitution. Given an expression e and an environment η such that fv(e) ⊆ dom(η), we define [e]₀ ∈ V as follows. When e is a quoted command ‘C’ we let [e]₀ = [‘C’]₀ = η(C). When e is an arithmetic expression, [e]₀ is defined in the expected way, except that arithmetic operations on quoted commands are, for definiteness, given the meaning 0. Thence we avoid the complications of introducing undefined expressions in a Hoare-style logic.¹⁰

A small-step operational semantics for the language is defined in Figure 11. A configuration (C, h) of the semantics consists of a closed command C and a heap h. An aborting configuration indicates a memory fault or a runtime “type error” due to confusion between integers and quoted commands.

Example D.1 (Iteration). The language does not include any high-level constructs for iteration. One can encode a “while” loop by means of “Landin’s knot” in the heap:

\[ \text{while } [e] \neq 0 \text{ do } C \quad \text{def} \]

\[ \begin{align*}
(\{ x \} \mapsto \text{new}(\text{skip}) & ) \in \\
\text{let } x = [e] & \text{ in} \\
(\text{if } (y = 0) & \text{ then free } x \\
\text{else } (C; \text{eval } [x]) & ) \\
\text{eval } [x] \\
\end{align*} \]

(Here x, y ∉ fv(e, C).) With that abbreviation, the following rule is derivable in the logic we present below:

\[ \Gamma \vdash \{ y \mapsto (y + 1) \} C \vdash \{ y \mapsto (y + 1) \} \]

D.2 Logic

The formulas of the logic [36] are called assertions and are generated by the grammar:
\[ P, Q ::= \text{false } \mid \text{true } \mid P \land Q \mid P \lor Q \mid P \Rightarrow Q \mid \forall x.P \mid \exists x.P \mid \text{int}(e) \mid e_1 = e_2 \mid e_1 \leq e_2 \mid e_1 \mapsto e_2 \mid \text{emp } \mid P \land Q \mid P \lor Q \]
\[ \{ P \} e \{ Q \} \mid P \land Q \mid \ldots \]

where the dots refer to atomic predicates and recursively defined predicates of the form \((\lambda n. P(n))\alpha)\) with α in P only occurring in “contractive” positions. (For space reasons, we do not formalize recursively defined assertions syntactically, but just treat them semantically, see below.) Unlike in standard separation logic, assertions are used both to describe predicates on heaps and to describe specifications of commands.

Indeed, the assertion \{ P \} e \{ Q \} means, intuitively, that the value of e is a quoted command ‘C’ which satisfies the Hoare triple with precondition P and postcondition Q in the usual sense of separation logic. Since Hoare triples are assertions, they can appear in pre- and post-conditions of other triples. Such nested triples are useful for reasoning about stored code: the specification of a command C can depend on the specification of other code in the heap, e.g.,

\[ \{ P \land \exists y. x \mapsto y \land \{ P' \} y(\{ Q' \}) \} C' \{ Q \} \]

Here a part of the precondition of C is that x points to a command y satisfying \{ P' \} y(\{ Q' \}). Presumably, the reason is that C contains one or more occurrences of eval[x].

The assertion P ⊗ Q should be thought of as “the assertion P extended with the invariant Q” and this assertion form is used to codify higher-order frame rules [15]. See [36] for detailed discussion of soundness and unsoundness of variations higher-order frame rules in the presence of higher-order store.

Proof Rules

The proof rules include the standard rules for intuitionalistic predicate logic and the logic of bunched implications [30]. Moreover, there are variations of standard separation logic proof rules (for dereferencing, sequencing, and so on). The proof rules can be found in Figure 12. Here Ω ranges over finite sets of variables.

Rule (⊗-FRAME) is a deep frame rule in which the invariant Q intuitively is added to all pre- and post-conditions inside P. The latter intuition is captured by the axioms in Figure 13. Rule (◦-FRAME) is a shallow (first-order) frame axiom. Finally, rule (EVAL) is the rule for executing stored code. Here, e → R[x] is an abbreviation of \exists x. e → x & R[x] (for an x not free in R).

D.3 A step-indexed model

To model invariant extension P ⊗ Q, Schwinghammer et. al. [36] models an assertion as a function that takes the meaning of a second, arbitrary assertion (to be thought of as the “invariant” that the first assertion is extended with) and gives a predicate on heaps.¹¹ This approach introduces a circularity, however, since such a function will in particular be applicable to itself. In the next section we show how to formalize and solve the circularity using metric spaces.

D.3.1 Semantic predicates

Following Section 2.3, we let UPred(H) be the set of subsets of \( \mathbb{N} \times H \) that are downwards closed in the first component:

\[ \{ p \subseteq \mathbb{N} \times H \mid \forall (k, h) \in p, \forall j \leq k. (j, h) \in p \}. \]

¹⁰ This idea follows earlier work on invariant extension [15, 16], which does not, however, deal with nested Hoare triples.
We give $UPred(H)$ the same distance function as in Section 2.3; the set then becomes a complete, bounded ultrametric space. Using Theorem 2.1 we obtain a unique $W \in \text{CBU}_{\text{HC}}$ satisfying

$$W \cong \frac{1}{2}(W \rightarrow UPred(H)).$$

Define $Pred = \frac{1}{2}(W \rightarrow UPred(H))$ and let $i : Pred \rightarrow W$ be the isomorphism. We will model assertions as elements of $Pred$.

Let the letters $p$ and $q$ range over elements of $Pred$. We order the elements of $Pred$ pointwise:

$$p \leq q \iff \forall w \in W . p(w) \subseteq q(w)$$

**Lemma D.2.** With the ordering above and the following operations, $Pred$ is a complete BI-algebra [14]:

- $\text{emp}(w) = \{(n,[])| n \in \mathbb{N}\}$
- $\text{emp}(w) = \{(n, h)| \exists h_1, h_2, h = h_1 \cdot h_2$
- $\wedge (n, h_1) \in p(w) \wedge (n, h_2) \in q(w)$
- $p \rightarrow q)(w) = \{(m, h)| \forall m \leq n.$
- $((m, h), (m, h')) \in p(w) \wedge h \neq h'$
- $= (m, h \cdot h')(w) \subseteq q(w)$

The fact that $Pred$ is a complete BI algebra immediately gives us a sound interpretation of most of the assertions in the logic [14], but to interpret recursive predicates we also need to know that the operations are non-expansive:

**Lemma D.3.** The BI-algebra operations on $Pred$ given by the previous lemma are non-expansive:

$$*, \rightarrow, \wedge, \forall : Pred \times Pred \rightarrow Pred$$

$$(I \rightarrow) : (I \rightarrow) \rightarrow Pred.$$
D.3.2 Interpretation of invariant extension
To interpret invariant-extension assertions $P \otimes Q$, we need a operator $\otimes$ on the set of semantic predicates $Pred$. The most convenient way to specify $\otimes$ is to give a certain recursive equation that it must satisfy. Using the metric-space setup we can then prove that there exists a unique operator satisfying this specification, by an easy application of Banach’s fixed point theorem, as in [36].

**Proposition D.4.** There exists a unique function $\otimes : Pred \times W \rightarrow Pred$ in CBUlt satisfying

$$p \otimes w = \lambda w'. p(w \circ w')$$

where $\circ : W \times W \rightarrow W$ is given by

$$w_1 \circ w_2 = i((i^{-1}(w_1) \otimes w_2) \ast i^{-1}(w_2)).$$

Observe that this is here that we exploit that we have obtained a proper solution to the world equation (59) as a metric space such that we can now easily establish the existence of the recursively-defined $\otimes$-operation.

The basic properties of $\otimes$ and $\circ$ are conveniently summarized as follows:

**Proposition D.5.** 1. $(W, \circ, \text{emp})$ is a monoid.

2. The operator $\otimes$ is a monoid action of $W$ on $Pred$: for all $P \in Pred$ and $w_1, w_2 \in W$ we have $P \otimes \text{emp} = P$ and $(P \otimes w_1) \otimes w_2 = P \otimes (w_1 \circ w_2)$.

D.3.3 Interpretation of assertions
We next define a semantic interpretation of Hoare triples. To this end we let $\text{Safe}_m$ be the set of configurations in the operational semantics that are safe for $m$ reduction steps, that is, those configurations that do not reduce to abort in $m$ (or fewer) steps. We write $\sim_k$ for the $k$-step reduction relation of the operational semantics.

Now say that $w \models_n (p,C,q)$ holds iff: For all $r \in UPred$, all $m < n$ and all heaps $h$, if $(m,h) \in p(w) \ast i^{-1}(w)(\text{emp}) \circ r$, then:

1. $(C,h) \in \text{Safe}_m$.

2. For all $k \leq m$ and all $h' \in H$, if $(C,h) \sim_k (\text{skip},h')$, then $(m-k,h') \in q(w) \ast i^{-1}(w)(\text{emp}) \circ r$.

This definition is similar to the one in [36] with its use of the invariant $w$ and the looping-in of the first order frame rule, i.e., the quantification over $r$. The difference is that the meaning is now relative to the operational semantics (rather than denotational) and that we use step-indexing to measure to what extent pre- and post-conditions should hold.

The intuition is, of course, that a Hoare-triple assertion is interpreted using the above semantic construct. However, to see that this interpretation gives a well-defined member of $Pred$, we need to know that a semantic Hoare triple is “non-expansive in $w$”:

**Proposition D.6.** If $w \models_k w'$ and $w \models_n (p,C,q)$, then $w' \models_{n\wedge(k-1)} (p,C,q)$.

**Proof.** Easy verification, using the fact that the separating conjunction $\ast$ on $Upred(V)$ is non-expansive (Lemma D.3). \qed

The interpretation of an assertion $\Gamma \vdash P \rightarrow Q$ is now defined to be an element $\llbracket P \rrbracket_\eta$, in $Pred$, for $\eta$ an environment mapping the variables in the domain of $\Gamma$ to $V$. The definition uses the complete BI-algebra structure on $Pred$ given earlier to interpret the standard logical connectives, e.g.,

$$\llbracket P \ast Q \rrbracket_\eta = \llbracket P \rrbracket_\eta \ast \llbracket Q \rrbracket_\eta.$$

Invariant extension is interpreted as follows:

$$\llbracket P \otimes Q \rrbracket_\eta = \left(\llbracket P \rrbracket_\eta \otimes i(\llbracket Q \rrbracket_\eta)\right) w$$

and, finally, Hoare triples are interpreted like this:

$$\llbracket [P;e;Q]_\eta \rrbracket = \begin{cases} \{ (n,h) \mid w \models_n ([P]_\eta \circ [Q]_\eta) \} & \text{if } \llbracket e \rrbracket_\eta \equiv \top \\ \emptyset & \text{otherwise.} \end{cases}$$

The concrete interpretation of all the logical connectives can be found in Figure 14. As in [36], recursively defined predicates are interpreted via Banach’s fixed point theorem:

**Proposition D.7.** Let $I$ be a set and suppose that, for each $i \in I$, $F_i : Pred^I \rightarrow Pred$ is a contractive function. Then there exists a unique $\bar{p} = (p_i)_{i \in I} \in Pred^I$ such that $F_i(\bar{p}) = p_i$, for all $i \in I$.

D.3.4 Soundness of proof rules
We define semantic validity of (open) assertions as follows: For an assertion $P$ with free variables belonging to $Γ$, say that $Γ \models P$. For all environment $η$ with $Γ \subseteq \text{dom}(η)$ and all $w \in W$ we have $\llbracket P \rrbracket_\eta = N \times H$. This amounts to saying that $\llbracket P \rrbracket_\eta$ is the top element of the BI algebra $Pred$.

**Theorem D.8.** If $Γ \vdash P$, then $Γ \models P$.

**Proof.** By showing the stronger property that each proof rule holds semantically, that is, with $\models$ replaced by $\models$. We only include the proof case for $\text{eval} [e]$ (the other interesting cases are the ones for invariant extension; there one uses Proposition D.5). We must show: if $Γ, z \models \llbracket e \rrbracket = \llbracket P \ast \text{eval} [e] \ast \{Q\} \rrbracket_\eta$, then $Γ \models \llbracket P \ast e \rightarrow \llbracket \bot \rrbracket \ast \{Q\} \rrbracket_\eta$.

Let $η$ be an environment with $Γ \subseteq \text{dom}(η)$, and let $w$ and $n$ be arbitrary. We must show that

$$w \models_n \{\llbracket P \ast \text{eval} [e] \ast \{Q\} \rrbracket \rrbracket_\eta \} \equiv \llbracket Q \rrbracket_\eta. \quad (60)$$

So let $k < n$ and $r \in UPred$ and let $(h,k) \in \llbracket P \ast \text{eval} [e] \ast \{Q\} \rrbracket_\eta (w) \ast i^{-1}(w)(\text{emp}) \circ r$. Then $h = h_1 \ast [l \rightarrow v] \ast h_2 \ast h_3$, where $(k,h_1) \in \llbracket P \rrbracket_\eta (w)$ and $[e]_\eta = l$ and $(k,l \rightarrow v) \in \llbracket R[z] \rrbracket (v) \ast w (h_2)$, and $(h_2) \in i^{-1}(w)(\text{emp})$ and $(h_3,h) \in r$. Using validity of the premise, we get that $(k,l \rightarrow v) \in \llbracket P \ast e \rightarrow \llbracket \bot \rrbracket \ast \{Q\} \rrbracket_\eta (v) \ast w (h_2)$, which means that $v = \top$ for some $C$, and that $w \models_{k} \llbracket P \ast e \rightarrow \llbracket \bot \rrbracket \ast \{C \rrbracket_\eta (v) \ast w (h_2) \ast r$. Now, if $k = 0$, then conditions 1 and 2 in the definition of $\models$ are clearly satisfied (item 2 because $(η(\text{eval} [e]),h)$ takes a reduction step), so (60) holds, as required. If $k > 0$ then, first observe that by downwards closure we have $(k-1,h) \in \llbracket P \ast \text{eval} [e] \ast \{Q\} \rrbracket_\eta (w) \ast i^{-1}(w)(\text{emp}) \circ r$. Therefore, $(C,h) \in \text{Safe}_{k-1}$, which implies that $\llbracket η(\text{eval} [e]),h \rrbracket \models_{k-1} (\text{skip},h')$ for some $h'$ and $m \leq k$. Then $(C,h) \sim_{m-1} (\text{skip},h')$. Since $m - 1 \leq k - 1$, we then get $(k-1) - (m - 1), (h') \in \llbracket Q \rrbracket_\eta (w) \ast i^{-1}(w)(\text{emp}) \circ r$, as required. \qed

D.4 Discussion
In summary, we have developed a new step-indexed model of separation logic with nested Hoare triples for reasoning about higher-order store. The new model is arguably simpler than the one in [36], since it is phrased directly in terms of the operational semantics without passing through a domain-theoretic denotational semantics. A usual advantage of domain-theory is a more abstract semantics, but in [36], it was in necessary to employ certain “step-like,” rank functions, so in the end the model of loc.cat. was not more abstract than the new one presented here.
|$\text{false}$|$_w$ = $\emptyset$
|$\text{true}$|$_w$ = $\mathbb{N} \times H$

|$P \land Q$|$_w$ = $|[P]|_w \cap |[Q]|_w$

|$P \lor Q$|$_w$ = $|[P]|_w \cup |[Q]|_w$

|$P \Rightarrow Q$|$_w$ = $\{ (n, h) \mid \forall m \leq n. (m, h) \in |[P]|_w \Rightarrow (m, h) \in |[Q]|_w \}$

|$\forall x. P$|$_w$ = $\bigcap_{v \in V} |[P]|_w$[\(x \mapsto v]\]$_w$

|$\exists x. P$|$_w$ = $\bigcup_{v \in V} |[P]|_w$[\(x \mapsto v]\]$_w$

|$\text{int}(e)$|$_w$ = \begin{cases} 
\mathbb{N} \times H & \text{if } \| e \|_\eta = m \text{ for some } m \in \mathbb{Z} \\
\emptyset & \text{otherwise}
\end{cases}

|$e_1 = e_2$|$_w$ = \begin{cases} 
\mathbb{N} \times H & \text{if } \| e_1 \|_\eta = \| e_2 \|_\eta \\
\emptyset & \text{otherwise}
\end{cases}

|$e_1 \leq e_2$|$_w$ = \begin{cases} 
\mathbb{N} \times H & \text{if } \| e_1 \|_\eta = m_1 \text{ and } \| e_2 \|_\eta = m_2 \text{ where } m_1 \leq m_2 \\
\emptyset & \text{otherwise}
\end{cases}

|$e_1 \mapsto e_2$|$_w$ = \begin{cases} 
\{ (n, |m \mapsto \| e_2 \|_\eta|) \mid n \in \mathbb{N} \} & \text{if } \| e_1 \|_\eta = m \text{ for some } m \in \mathbb{Z} \\
\emptyset & \text{otherwise}
\end{cases}

|$\text{emp}$|$_w$ = $\mathbb{N} \times \{ () \}$

|$P \ast Q$|$_w$ = $|[P]|_w \ast |[Q]|_w$

|$P \rightarrow Q$|$_w$ = $|[P]|_w \rightarrow |[Q]|_w$

|$\{ P \ast \{ Q \} \}$|$_w$ = \begin{cases} 
\{ (n, h) \mid w \models_n (|P|_\eta, C, |[Q]|_\eta) \} & \text{if } \| e \|_\eta = 'C' \\
\emptyset & \text{otherwise}
\end{cases}

|$P \otimes Q$|$_w$ = $\left( |[P]|_\eta \otimes i([Q]|_\eta) \right)_w$

---

Figure 14. Interpretation of assertions.