Realizability Semantics of Parametric Polymorphism, General References, and Recursive Types

Lars Birkedal, Kristian Støvring, and Jacob Thamsborg

IT University of Copenhagen*

Abstract. We present a realizability model for a call-by-value, higher-order programming language with parametric polymorphism, general first-class references, and recursive types. The main novelty is a relational interpretation of open types (as needed for parametricity reasoning) that include general reference types. The interpretation uses a new approach to modeling references.

The universe of semantic types consists of world-indexed families of logical relations over a universal predomain. In order to model general reference types, worlds are finite maps from locations to semantic types: this introduces a circularity between semantic types and worlds that precludes a direct definition of either. Our solution is to solve a recursive equation in an appropriate category of metric spaces. In effect, types are interpreted using a Kripke logical relation over a recursively defined set of worlds.

We illustrate how the model can be used to prove simple equivalences between different implementations of imperative abstract data types.

1 Introduction

In this article we develop a semantic model of a call-by-value programming language with impredicative and parametric polymorphism, general first-class references, and recursive types. Motivations for conducting this study include:

- Extending the approach to reasoning about abstract data types via relational parametricity from pure languages to more realistic languages with effects, here general references. We discussed this point of view extensively earlier [8].
- Investigating what semantic structures are needed in general models for effects. Indeed, we see the present work as a pilot study for studying general type theories and models of effects (e.g., [12, 19]), in which we identify key ingredients needed for semantic modeling of general first-class references.
- Paving the way for developing models of separation logic for ML-like languages with reference types. Earlier such models of separation logic [16] only treat so-called strong references, where the type on the contents of a reference cell can vary: therefore proof rules cannot take advantage of the strong invariants provided by ML-style reference types.

* Rued Langgaards Vej 7, 2300 Copenhagen S, Denmark

We now give an overview of the conceptual development of the paper. The development is centered around three recursively defined structures, defined in three stages. In slogan form, there is one recursively defined structure for each of the type constructors $\forall$, ref, and μ alluded to in the title.

First, since the language involves impredicative polymorphism, the semantic model is based on a realizability interpretation [4] over a certain recursively defined predomain V. Using this predomain we can give a denotational semantics of an untyped version of the language. This part is mostly standard, except for the fact that we model locations as pairs (l, n), with l a natural number corresponding to a standard location and $n \in \mathbb{N} \cup \{\infty\}$ indicating the "approximation stage" of the location [8]. These pairs, called semantic locations, are needed for modeling reference types in stage three.[1]

Second, to account for dynamic allocation of typed reference cells, we follow earlier work on modeling simple integer references [7] and use a Kripke-style possible worlds model. Here, however, the set of worlds needs to be recursively defined since we treat general references. Semantically, a world maps locations to semantic types, which, following the general realizability idea, are certain world-indexed families of relations on V: this introduces a circularity between semantic types and worlds that precludes a direct definition of either. Thus we need to solve recursive equations of approximately the following form

$$\mathcal{W} = \mathbb{N}_0 \rightharpoonup_{fin} \mathcal{T}$$
$$\mathcal{T} = \mathcal{W} \to \mathit{CURel}(V)$$

even in order to define the space in which types will be modeled. We formally define the recursive equations in certain ultrametric spaces and show how to solve them using known results from metric-space based semantics. The employed metric on relations on V is well-known from work on interpreting recursive types and impredicative polymorphism [1, 4, 5, 10, 13]; here we extend its use to reference types (combined with these two other features).

Third, having now defined the space in which types should be modeled, the actual semantics of types can be defined. For recursive types, that also involves a recursive definition. Since the space $\mathcal{T}$ of semantic types is a metric space we can employ Banach's fixed point theorem to find a solution as the fixed point of a contractive operator on $\mathcal{T}$.[2] This involves interpreting the various type constructors of the language as non-expansive operators. For most type constructors doing so is straightforward, but for the reference-type constructor

[1] Intuitively, the problem with modeling locations using a flat cpo of natural numbers is that such "flat" locations contain no approximation information that can be used to define relations by induction. (See page 12.)

[2] We remark that the fixed point could also be found using the technique of Pitts [18]; the proof techniques are very similar because of the particular way the requisite metrics are defined. In this article we do in any case need the metric-space formulation, but not the extra separation of positive and negative arguments in recursive definitions of relations, and hence we define the meaning of recursive types via Banach's fixed point theorem.

it is not. That is the reason for introducing the semantic locations mentioned above: using these, we can define a semantic reference-type operator (and show that it is non-expansive).

Finally, having now defined semantics of types using a family of world-indexed logical relations, we define the typed meaning of terms by proving the fundamental theorem of logical relations wrt. the untyped semantics of terms.

In this article we do not consider operational semantics but focus on presenting the model outlined above. We have earlier shown a computational-adequacy result for a semantics similar to the untyped semantics defined in stage one [8]: we expect that result to carry over to the present setup. Also, the model does not validate standard equivalences involving local state,[3] although we expect that it can be extended to do so (see Section 6). Here we rather aim to present the fundamental ideas behind Kripke logical relations over recursively defined sets of worlds.

The remainder of the article is organized as follows. Section 2 sketches the language we consider. In Section 3 we present the untyped semantics, corresponding to stage one in the outline above. In Section 4 we present the typed semantics, corresponding to the last two stages. In Section 5 we present a few examples of reasoning using the model. Related work is discussed in Section 6.

Because of space limitations, some definitions and most proofs have been omitted from this article. They can be found in the long version, available from the authors' web pages.[4]

2 Language

We consider a standard call-by-value language with universal types, iso-recursive types, ML-style reference types, and a ground type of integers. The language is sketched in Figure 1. Terms are not intrinsically typed; this allows us to give a denotational semantics of untyped terms. The typing rules are standard [17]. In the figure, Ξ and Γ range over contexts of type variables and term variables, respectively. As we do not consider operational semantics in this article, there is no need for location constants, and hence no need for store typings.

3 Untyped Semantics

We now give a denotational semantics for the untyped term language above. As usual for models of untyped languages, the semantics is given by means of a "universal" complete partial order (cpo) in which one can inject integers, pairs, functions, etc. This universal cpo is obtained by solving a recursive predomain equation. The only non-standard aspect of the semantics is the treatment of store locations: locations are modeled as elements of the cpo $Loc = \mathbb{N}_0 \times \overline{\omega}$ where $\overline{\omega}$ is the "vertical natural numbers" cpo: $1 \sqsubset 2 \sqsubset \cdots \sqsubset n \sqsubset \cdots \sqsubset \infty$. (For notational reasons it is convenient to call the least element 1 rather than 0.) The

[3] The model can only equate computations that allocate references "in lockstep".

[4] Currently: `http://www.itu.dk/people/kss/papers/poly-ref-rec.pdf`

$$\text{Types:} \quad \tau ::= \mathsf{int} \mid 1 \mid \tau_1 \times \tau_2 \mid \tau_1 + \tau_2 \mid \tau_1 \to \tau_2 \mid \mu\alpha.\tau \mid \forall\alpha.\tau \mid \alpha \mid \mathsf{ref}\,\tau$$

$$\begin{aligned}\text{Terms:} \quad t ::= {} & x \mid \overline{n} \mid \mathsf{ifz}\, t_0\, t_1\, t_2 \mid t_1 + t_2 \mid t_1 - t_2 \mid () \mid (t_1, t_2) \mid \mathsf{fst}\, t \mid \mathsf{snd}\, t \\ & \mid \mathsf{inl}\, t \mid \mathsf{inr}\, t \mid \mathsf{case}\, t_0\, x_1.t_1\, x_2.t_2 \mid \lambda x.t \mid \mathsf{fix}\, f.\lambda x.t \mid t_1\, t_2 \\ & \mid \mathsf{fold}\, t \mid \mathsf{unfold}\, t \mid \Lambda\alpha.t \mid t\,[\tau] \mid \mathsf{ref}\, t \mid !t \mid t_1 := t_2\end{aligned}$$

Sample typing rules:

$$\frac{\Xi \mid \Gamma \vdash t : \tau[\mu\alpha.\tau/\alpha]}{\Xi \mid \Gamma \vdash \mathsf{fold}\, t : \mu\alpha.\tau} \qquad \frac{\Xi \mid \Gamma \vdash t : \mu\alpha.\tau}{\Xi \mid \Gamma \vdash \mathsf{unfold}\, t : \tau[\mu\alpha.\tau/\alpha]}$$

$$\frac{\Xi, \alpha \mid \Gamma \vdash t : \tau}{\Xi \mid \Gamma \vdash \Lambda\alpha.t : \forall\alpha.\tau}\ (\Xi \vdash \Gamma) \qquad \frac{\Xi \mid \Gamma \vdash t : \forall\alpha.\tau_0}{\Xi \mid \Gamma \vdash t\,[\tau_1] : \tau_0[\tau_1/\alpha]}\ (\Xi \vdash \tau_1)$$

$$\frac{\Xi \mid \Gamma \vdash t : \tau}{\Xi \mid \Gamma \vdash \mathsf{ref}\, t : \mathsf{ref}\,\tau} \qquad \frac{\Xi \mid \Gamma \vdash t : \mathsf{ref}\,\tau}{\Xi \mid \Gamma \vdash !t : \tau}$$

$$\frac{\Xi \mid \Gamma \vdash t_1 : \mathsf{ref}\,\tau \qquad \Xi \mid \Gamma \vdash t_2 : \tau}{\Xi \mid \Gamma \vdash t_1 := t_2 : 1}$$

Fig. 1. Programming language

intuitive idea is that locations can be approximated: the element $(l, \infty) \in \mathit{Loc}$ is the "ideal" location numbered l, while the elements of the form (l, n) for $n \neq \infty$ are its approximations. It is essential for the construction of the typed semantics (in the next section) that these "approximate locations" (l, n) are included.

Let in the following Cpo be the category of ω-cpos and ω-continuous functions. We use the standard notation for products, sums, lifting, and function spaces in Cpo. Injections into binary sums are written ι_1 and ι_2. For any set M and any cpo A, the cpo $M \rightharpoonup_{\mathit{fin}} A$ has maps from finite subsets of M to A as elements, and is ordered as follows: $f \sqsubseteq f'$ if and only if f and f' has the same domain M_0 and $f(m) \sqsubseteq f'(m)$ for all $m \in M_0$. The Kleisli composition $g \mathbin{\overline{\circ}} f$ of two continuous functions $f : A \to B_\perp$ and $g : B \to C_\perp$ is given by $(g \mathbin{\overline{\circ}} f)(a) = g\, b$ if $f\, a = \lfloor b \rfloor$ for some b, and $(g \mathbin{\overline{\circ}} f)(a) = \perp$ otherwise. A complete, pointed partial order (cppo) is a cpo containing a least element. The least fixed-point of a continuous function $f : D \to D$ from a cppo D to itself is written $\mathit{fix}\, f$.

The semantics below is presented in monadic style [15], i.e., structured using a monad that models the effects of the language. It is most convenient to define this monad by means of a Kleisli triple: for every cpo S and every cppo Ans, the continuation-and-state monad $T_{S,\mathit{Ans}} : \mathsf{Cpo} \to \mathsf{Cpo}$ over S and Ans is given by $T_{S,\mathit{Ans}}\, A = (A \to S \to \mathit{Ans}) \to S \to \mathit{Ans}$, $\eta_A\, a = \lambda k.\lambda s.\, k\, a\, s$, and $c \ \star_{A,B}\ f = \lambda k.\lambda s.\, c\, (\lambda a.\lambda s'.f\, a\, k\, s')\, s$ (with $\eta_A : A \to T_{S,\mathit{Ans}} A$ and $\star_{A,B} : T_{S,\mathit{Ans}} A \to (A \to T_{S,\mathit{Ans}} B) \to T_{S,\mathit{Ans}} B$.) In the following we omit the type subscripts on η and $\star$.[5]

[5] Continuations are included for a technical reason, namely to ensure chain-completeness of the relations that will be used to model computations.

The standard methods for solving recursive (pre)domain equations give solutions that satisfy certain induction principles [18, 21]. One aspect of these induction principles is that, loosely speaking, one obtains as a solution not only a cpo A, but also a family of "projection" functions ϖ_n on A (one function for each $n \in \omega$) such that each element a of A is the limit of its projections $\varpi_0(a)$, $\varpi_1(a)$, etc. These functions therefore provide a handle for proving properties about A by induction on n.

Definition 1. *A* uniform cpo $(A, (\varpi_n)_{n\in\omega})$ *is a cpo* A *together with a family* $(\varpi_n)_{n\in\omega}$ *of continuous functions from* A *to* $A_\perp$*, satisfying*

$$\begin{aligned} \varpi_0 &= \lambda e.\, \bot \\ \varpi_0 &\sqsubseteq \varpi_1 \sqsubseteq \cdots \sqsubseteq \varpi_n \sqsubseteq \ldots \\ \bigsqcup_{n\in\omega} \varpi_n &= \lambda a.\lfloor a \rfloor \\ \varpi_m \mathbin{\overline{\circ}} \varpi_n &= \varpi_n \mathbin{\overline{\circ}} \varpi_m = \varpi_{\min(m,n)}\,. \end{aligned}$$

We are now ready to construct a uniform cpo $(V, (\pi_n)_{n\in\omega})$ such that V is a suitable "universal" cpo. The functions π_n will be used in the definition of the untyped semantics. Intuitively, if one for example looks up the approximate location $(l,\ n{+}1)$ in a store s, one only obtains the approximate element $\pi_n(s(l))$ as a result.

Proposition 2. *There exists a uniform cpo* $(V, (\pi_n)_{n\in\omega})$ *satisfying the following two properties:*

1. *The following isomorphism holds in* Cpo*:*

$$V \;\cong\; \mathbb{Z} + \mathit{Loc} + 1 + (V \times V) + (V + V) + (V \to T_{S,\mathit{Ans}}V) + V + T_{S,\mathit{Ans}}V \quad (1)$$

   where $T_{S,\mathit{Ans}}V = (V \to S \to \mathit{Ans}) \to S \to \mathit{Ans}$*,* $S = \mathbb{N}_0 \rightharpoonup_{\mathit{fin}} V$*,* $\mathit{Ans} = (\mathbb{Z} + \mathit{Err})_\perp$*,* $\mathit{Loc} = \mathbb{N}_0 \times \overline{\omega}$*, and* $\mathit{Err} = 1$*.*
2. *Abbreviate* $TV = T_{S,\mathit{Ans}}V$ *and* $K = V \to S \to \mathit{Ans}$*. Define the following injection functions corresponding to the summands on the right-hand side of the isomorphism* (1)*:*

$$\begin{aligned} \mathit{in}_{\mathbb{Z}} &: \mathbb{Z} \to V & \mathit{in}_{+} &: V + V \to V \\ \mathit{in}_{\mathit{Loc}} &: \mathit{Loc} \to V & \mathit{in}_{\to} &: (V \to TV) \to V \\ \mathit{in}_{1} &: 1 \to V & \mathit{in}_{\mu} &: V \to V \\ \mathit{in}_{\times} &: V \times V \to V & \mathit{in}_{\forall} &: TV \to V \end{aligned}$$

   With that notation, the functions $\pi_n : V \to V_\perp$ *satisfy (and are determined by) the equations shown in Figure 2.*

These two properties determine V *uniquely, up to isomorphism in* Cpo*.*

$$
\begin{aligned}
\pi_0 &= \lambda v.\bot \\
\pi_{n+1}(in_{\mathbb{Z}}(m)) &= \lfloor in_{\mathbb{Z}}(m) \rfloor \\
\pi_{n+1}(in_1(*)) &= \lfloor in_1(*) \rfloor \\
\pi_{n+1}(in_{Loc}(l,\infty)) &= \lfloor in_{Loc}(l, n+1) \rfloor \\
\pi_{n+1}(in_{Loc}(l,m)) &= \lfloor in_{Loc}(l, \min(n+1,m)) \rfloor \\
\pi_{n+1}(in_{\times}(v_1,v_2)) &= \begin{cases} \lfloor in_{\times}(v_1',v_2') \rfloor & \text{if } \pi_n\, v_1 = \lfloor v_1' \rfloor \text{ and } \pi_n\, v_2 = \lfloor v_2' \rfloor \\ \bot & \text{otherwise} \end{cases} \\
\pi_{n+1}(in_{+}(\iota_i\, v)) &= \begin{cases} \lfloor in_{+}(\iota_i\, v') \rfloor & \text{if } \pi_n\, v = \lfloor v' \rfloor \\ \bot & \text{otherwise} \end{cases} \qquad (i = 1,2) \\
\pi_{n+1}(in_{\mu}\, v) &= \begin{cases} \lfloor in_{\mu}\, v' \rfloor & \text{if } \pi_n v = \lfloor v' \rfloor \\ \bot & \text{otherwise} \end{cases} \\
\pi_{n+1}(in_{\forall}\, c) &= \lfloor in_{\forall}(\pi_{n+1}^T\, c) \rfloor \\
\pi_{n+1}(in_{\rightarrow}\, f) &= \left\lfloor in_{\rightarrow}\left(\lambda v.\, \begin{cases} \pi_{n+1}^T\,(f\, v') & \text{if } \pi_n\, v = \lfloor v' \rfloor \\ \bot & \text{otherwise} \end{cases}\right)\right\rfloor
\end{aligned}
$$

Here the functions $\pi_n^S : S \to S_\bot$ and $\pi_n^K : K \to K$ and $\pi_n^T : TV \to TV$ are defined as follows:

$$
\begin{aligned}
\pi_0^S &= \lambda s.\bot \qquad \pi_0^K = \lambda k.\bot \qquad \pi_0^T = \lambda c.\bot \\
\pi_{n+1}^S(s) &= \begin{cases} \lfloor s' \rfloor & \text{if } \pi_n \circ s = \lambda l.\lfloor s'(l) \rfloor \\ \bot & \text{otherwise} \end{cases} \\
\pi_{n+1}^K(k) &= \lambda v.\lambda s.\begin{cases} k\, v'\, s' & \text{if } \pi_n\, v = \lfloor v' \rfloor \text{ and } \pi_{n+1}^S\, s = \lfloor s' \rfloor \\ \bot & \text{otherwise} \end{cases} \\
\pi_{n+1}^T(c) &= \lambda k.\lambda s.\begin{cases} c\,(\pi_{n+1}^K\, k)\, s' & \text{if } \pi_{n+1}^S\, s = \lfloor s' \rfloor \\ \bot & \text{otherwise}\,. \end{cases}
\end{aligned}
$$

Fig. 2. Characterization of the projection functions $\pi_n : V \to V_\bot$.

Proof (sketch). By modifying the usual projection functions, obtained from a minimal-invariant solution of (1), on arguments corresponding to locations. □

From here on, let V and $(\pi_n)_{n\in\omega}$ be as in the proposition above. We furthermore use the abbreviations, notation for injections, etc. introduced in the proposition; in particular, $TV = (V \to S \to Ans) \to S \to Ans$. Additionally, abbreviate $\lambda_l = in_{Loc}(l,\infty)$ and $\lambda_l^n = in_{Loc}(l,n)$. Let $error_{Ans} = \lfloor \iota_2 * \rfloor \in Ans$ be the "error answer" and let $error = \lambda k.\lambda s.\, error_{Ans} \in TV$ be the "error computation".

In order to model the three operations of the untyped language that involve references, we define the three functions $alloc : V \to TV$, $lookup : V \to TV$, and $assign : V \to V \to TV$. The first two of these functions are shown in the lower part of Figure 3. The third function, *assign*, is similar to *lookup*: the idea is that when one assigns a value to an approximate location, only an approximate value is inserted in the store. Notice that it would not suffice to

For every t with $\mathrm{FV}(t) \subseteq X$, define the continuous $[\![t]\!]_X : V^X \to TV$ by induction on t:

$$
\begin{aligned}
[\![x]\!]_X\,\rho &= \eta(\rho(x)) \\
[\![\lambda x.t]\!]_X\,\rho &= \eta(in_{\to}(\lambda v.\ [\![t]\!]_{X,x}\,(\rho[x \mapsto v]))) \\
[\![t_1\ t_2]\!]_X\,\rho &= [\![t_1]\!]_X\,\rho \star \lambda v_1.\ [\![t_2]\!]_X\,\rho \star \lambda v_2.\ \begin{cases} f\ v_2 & \text{if } v_1 = in_{\to}\ f \\ \mathit{error} & \text{otherwise} \end{cases} \\
[\![\Lambda\alpha.t]\!]_X\,\rho &= \eta(in_{\forall}\,([\![t]\!]_X\,\rho)) \\
[\![t\ [\tau]]\!]_X\,\rho &= [\![t]\!]_X\,\rho \star \lambda v.\ \begin{cases} c & \text{if } v = in_{\forall}\,c \\ \mathit{error} & \text{otherwise} \end{cases} \\
[\![\mathsf{ref}\ t]\!]_X\,\rho &= [\![t]\!]_X\,\rho \star \lambda v.\ \mathit{alloc}\,v \\
[\![!\,t]\!]_X\,\rho &= [\![t]\!]_X\,\rho \star \lambda v.\ \mathit{lookup}\,v \\
[\![t_1 := t_2]\!]_X\,\rho &= [\![t_1]\!]_X\,\rho \star \lambda v_1.\ [\![t_2]\!]_X\,\rho \star \lambda v_2.\ \mathit{assign}\ v_1\ v_2 \\
&\cdots
\end{aligned}
$$

$$
\mathit{alloc}\,v = \lambda k\,\lambda s.\ k\ (\lambda_{\mathit{free}(s)})\ (s[\mathit{free}(s) \mapsto v])
$$

$$
(\text{where } \mathit{free}(s) = \min\{n \in \mathbb{N}_0 \mid n \notin \mathrm{dom}(s)\})
$$

$$
\mathit{lookup}\,v = \lambda k\,\lambda s.\ \begin{cases} k\ s(l)\ s & \text{if } v = \lambda_l \text{ and } l \in \mathrm{dom}(s) \\ k\ v'\ s & \text{if } v = \lambda_l^{n+1},\ l \in \mathrm{dom}(s), \text{ and } \pi_n(s(l)) = \lfloor v' \rfloor \\ \bot_{Ans} & \text{if } v = \lambda_l^{n+1},\ l \in \mathrm{dom}(s), \text{ and } \pi_n(s(l)) = \bot \\ \mathit{error}_{Ans} & \text{otherwise} \end{cases}
$$

Fig. 3. Untyped semantics of terms (sample cases).

define, e.g., $\mathit{lookup}(\lambda_l^{n+1})(k)(s) = \bot$ for $l \in \mathrm{dom}(s)$, and hence avoid mentioning the projection functions: *lookup* would then not be continuous.

We are now ready to define the untyped semantics.

Definition 3. *Let t be a term and let X be a set of term variables such that $\mathrm{FV}(t) \subseteq X$. The* untyped semantics of t with respect to X *is the continuous function $[\![t]\!]_X : V^X \to TV$ defined by induction on t in Figure 3.*

The semantics of a complete program is defined by supplying an initial continuation and the empty store:

Definition 4. *Let t be a term with no free term variables or type variables. The* program semantics of t *is the element $[\![t]\!]^{\mathrm{p}}$ of Ans defined by $[\![t]\!]^{\mathrm{p}} = [\![t]\!]_\emptyset\,\emptyset\,k_{init}\,s_{init}$ where $s_{init} \in S$ is the empty store and*

$$
k_{init} = \lambda v.\lambda s.\ \begin{cases} \lfloor \iota_1\,m \rfloor & \text{if } v = in_{\mathbb{Z}}(m) \\ \mathit{error}_{Ans} & \text{otherwise.} \end{cases}
$$

We emphasize that even though the above semantics is slightly non-standard because of the use of the projection functions in lookup and assignment, we can still use it to reason about operational behaviour: as mentioned in the Introduction an earlier adequacy proof [8] should carry over to the present setting.

4 Typed Semantics

In this section we present a "typed semantics", i.e., an interpretation of types and typed terms. As described in the introduction, types will be interpreted as world-indexed families of binary relations on the universal cpo V. Since worlds depend on semantic types, the space of semantic types is obtained by solving a recursive metric-space equation, i.e., by finding a fixed-point of a functor on metric spaces.

4.1 Ultrametric Spaces

Recall that a metric space (X, d) is 1-*bounded* if $d(x, y) \leq 1$ for all x and y in X. Let CBUlt be the category with complete, 1-bounded ultrametric spaces as objects and non-expansive (i.e., non-distance-increasing) functions as morphisms [6]. This category is cartesian closed [22]; here one needs the ultrametric inequality. The exponential $(X_1, d_1) \to (X_2, d_2)$ has the set of non-expansive functions from (X_1, d_1) to (X_2, d_2) as the underlying set, and the "sup"-metric $d_{X_1 \to X_2}$ as distance function: $d_{X_1 \to X_2}(f, g) = \sup\{d_2(f(x), g(x)) \mid x \in X_1\}$.

For a given non-empty, complete metric space, consider the function *fix* that maps every contractive operator to its unique fixed-point (which exists by Banach's fixed-point theorem). On complete ultrametric spaces, *fix* is non-expansive in the following sense.

Proposition 5. *Let $(X, d) \in \mathsf{CBUlt}$ be non-empty. For all contractive functions f and g from (X, d) to itself, $d(\mathit{fix}\, f, \mathit{fix}\, g) \leq d(f, g)$.*

4.2 The Space of Semantic Types

We now turn to constructing the space of semantic types. First, some standard definitions. For every cpo A, let $\mathit{Rel}(A)$ be the set of binary relations $R \subseteq A \times A$ on A. A relation $R \in \mathit{Rel}(A)$ is *complete* if for all chains $(a_n)_{n\in\omega}$ and $(a'_n)_{n\in\omega}$ such that $(a_n, a'_n) \in R$ for all n, also $(\sqcup_{n\in\omega} a_n, \sqcup_{n\in\omega} a'_n) \in R$. Let $\mathit{CRel}(A)$ be the set of complete relations on A. For every cpo A and every relation $R \in \mathit{Rel}(A)$, define the relation $R_\perp \in \mathit{Rel}(A_\perp)$ by $R_\perp = \{\, (\perp, \perp) \,\} \cup \{\, (\lfloor a \rfloor, \lfloor a' \rfloor) \mid (a, a') \in R \,\}$. For $R \in \mathit{Rel}(A)$ and $S \in \mathit{Rel}(B)$, let $R \to S$ be the set of continuous functions f from A to B satisfying that for all $(a, a') \in R$, $(f\, a, f\, a') \in S$.

On uniform cpos, we furthermore define *uniform* binary relations [1, 4]:

Definition 6. *Let $(A, (\varpi_n)_{n\in\omega})$ be a uniform cpo. A relation $R \in \mathit{Rel}(A)$ is* uniform with respect to $(\varpi_n)_{n\in\omega}$ *if $\varpi_n \in R \to R_\perp$ for all n. Let $\mathit{CURel}(A, (\varpi_n)_{n\in\omega})$ be the set of binary relations on A that are complete and uniform with respect to $(\varpi_n)_{n\in\omega}$.*

Below we define a number of metric spaces that will be used in the interpretation of types. After defining one of these metric spaces (X, d), the "distance function" d will be fixed, so we usually omit it and call X itself a metric space.

Let in the following $(A, (\varpi_n)_{n\in\omega})$ be a uniform cpo and let $\mathit{CURel}(A) = \mathit{CURel}(A, (\varpi_n)_{n\in\omega})$. First, as in Amadio [4], we obtain:

Proposition 7. *The set $CURel(A)$ is a complete, 1-bounded ultrametric space with the distance function given by*

$$d(R,S) = \begin{cases} 2^{-\max\{\, n\in\omega \;\mid\; \varpi_n\in R\to S_\perp \;\wedge\; \varpi_n\in S\to R_\perp \,\}} & \text{if } R \neq S \\ 0 & \text{if } R = S. \end{cases}$$

We also need metrics on "worlds" and monotone functions from worlds:

Proposition 8. *Let $(X,d) \in \mathsf{CBUlt}$. The set $\mathbb{N}_0 \rightharpoonup_{fin} X$ of finite maps from natural numbers to elements of X is a complete, 1-bounded ultrametric space with the distance function given by*

$$d'(\Delta,\Delta') = \begin{cases} \max\{d(\Delta(l),\Delta'(l)) \mid l \in \mathrm{dom}(\Delta)\} & \text{if } \mathrm{dom}(\Delta) = \mathrm{dom}(\Delta') \\ 1 & \text{otherwise.} \end{cases}$$

Definition 9. *For every $(X,d) \in \mathsf{CBUlt}$, define an "extension" ordering $\leq$ on $\mathbb{N}_0 \rightharpoonup_{fin} X$ by $\Delta \leq \Delta' \iff \mathrm{dom}(\Delta) \subseteq \mathrm{dom}(\Delta') \wedge \forall l \in \mathrm{dom}(\Delta).\, \Delta(l) = \Delta'(l)$.*

Proposition 10. *Let $(X,d) \in \mathsf{CBUlt}$, and let $(\mathbb{N}_0 \rightharpoonup_{fin} X) \to_{mon} CURel(A)$ be the set of functions ν from $\mathbb{N}_0 \rightharpoonup_{fin} X$ to $CURel(A)$ that are both non-expansive and monotone in the sense that $\Delta \leq \Delta'$ implies $\nu(\Delta) \subseteq \nu(\Delta')$. This set is a complete, 1-bounded ultrametric space with the "sup"-metric, given by*

$$d'(\nu,\nu') = \sup\{d(\nu(\Delta),\nu'(\Delta)) \mid \Delta \in \mathbb{N}_0 \rightharpoonup_{fin} X\}.$$

Proof (sketch). It suffices to show that the set of monotone and non-expansive functions is a closed subset of the (complete) metric space of all non-expansive functions. To that end, one needs the following property: if $R,S \in CURel(A)$ satisfy that $\varpi_n \in R \to S_\perp$ for all n, then $R \subseteq S$. □

In the rest of this section we do not need the extra generality of uniform cpos: recall that V is the cpo obtained from Proposition 2 and abbreviate $CURel(V) = CURel(V,(\pi_n)_{n\in\omega})$.

Proposition 11. *The operation mapping each $(X,d) \in \mathsf{CBUlt}$ to the metric space $(\mathbb{N}_0 \rightharpoonup_{fin} X) \to_{mon} CURel(V)$ (as given by the previous proposition) can be extended to a functor $F : \mathsf{CBUlt}^{\mathrm{op}} \to \mathsf{CBUlt}$ in the natural way.*

Given $(X,d) \in \mathsf{CBUlt}$ and $0 < \delta < 1$ one defines $\delta \cdot (X,d) \in \mathsf{CBUlt}$ with the same underlying set X but with all distances multiplied by δ.

Theorem 12. *There exists a unique (up to isomorphism) complete, 1-bounded ultrametric space $\widehat{\mathcal{T}}$ such that the following isomorphism holds in CBUlt:*

$$\widehat{\mathcal{T}} \cong \tfrac{1}{2}((\mathbb{N}_0 \rightharpoonup_{fin} \widehat{\mathcal{T}}) \to_{mon} CURel(V)). \tag{2}$$

Proof (sketch). By a well-known adaptation of the inverse-limit method [6, 20, 22] one can show that so-called locally contractive mixed-variance functors on CBUlt have unique fixed-points up to isomorphism. The functor F defined in the previous proposition is only locally non-expansive (i.e., non-expansive as a function on each hom-set) so we use the standard method of multiplying F with the "shrinking factor" $\delta = 1/2$, thus obtaining a locally contractive functor. □

4.3 Interpretation of Types

Let in the following $\widehat{\mathcal{T}}$ be a complete, 1-bounded ultrametric space satisfying (2), and let $App : \widehat{\mathcal{T}} \to \frac{1}{2}((\mathbb{N}_0 \rightharpoonup_{fin} \widehat{\mathcal{T}}) \to_{mon} CURel(V))$ be an isomorphism. For convenience, we use the following abbreviations (where the names $\mathcal{W}$ and $\mathcal{T}$ are intended to indicate "worlds" and "types", respectively):

$$\mathcal{W} = \mathbb{N}_0 \rightharpoonup_{fin} \widehat{\mathcal{T}}$$
$$\mathcal{T} = \mathcal{W} \to_{mon} CURel(V)\,.$$

With that notation, (2) expresses that $\widehat{\mathcal{T}}$ is isomorphic to $\frac{1}{2}\mathcal{T}$. We choose $\mathcal{T}$ as our space of semantic types.

We additionally define families of relations on "states" (elements of S), "continuations" (elements of $K = V \to S \to Ans$), and "computations" (elements of TV). First, $(S, (\pi_n^S)_{n\in\omega})$ is a uniform cpo; abbreviate $CURel(S) = CURel(S, (\pi_n^S)_{n\in\omega})$. We then define

$$\mathcal{T}_S = \mathcal{W} \to CURel(S)$$

as the element of CBUlt obtained from Proposition 7 and the exponential in CBUlt. (The elements of $\mathcal{T}_S$ are non-expansive but not necessarily monotone functions.) As for continuations and computations, one observes that $(K, (\pi_n^K)_{n\in\omega})$ and $(TV, (\pi_n^T)_{n\in\omega})$ are "uniform cppos", i.e., satisfy conditions similar to those in Definition 1, but in the category of cppos and strict continuous functions (see the long version of this article for the details). Using analogues of Definition 6 and Propositions 7 and 10 we obtain $CURel(K) = CURel(K, (\pi_n^K)_{n\in\omega})$ and $CURel(TV) = CURel(TV, (\pi_n^T)_{n\in\omega})$ in CBUlt and define

$$\mathcal{T}_K = \mathcal{W} \to_{mon} CURel(K)$$
$$\mathcal{T}_T = \mathcal{W} \to_{mon} CURel(TV)\,.$$

In all the ultrametric spaces we consider here, all non-zero distances have the form 2^{-m} for some m. For such ultrametric spaces, there is a useful notion of n-approximated equality of elements: the notation $x =_n y$ means that $d(x, y) \leq 2^{-n}$. The ultrametric inequality then amounts to the fact that each relation $=_n$ is transitive, and therefore an equivalence relation. The factor $1/2$ in (2) implies that worlds that are "$(n + 1)$-equal" only contain "n-equal" semantic types.

To interpret types of the language as elements of $\mathcal{T}$, it remains to define a number of operators on $\mathcal{T}$ (and $\mathcal{T}_T$ and $\mathcal{T}_K$) that will be used to interpret the various type constructors of the language; these operators are shown in Figure 4. Notice that the operator *ref* is defined in terms of n-approximated equality $=_n$ on $CURel(V)$. In order to interpret the fragment of the language without recursive types, it suffices to verify that these operators are well-defined (e.g., *ref* actually maps elements of $\mathcal{T}$ into $\mathcal{T}$.) In order to interpret recursive types, however, we furthermore need to verify that the operators are non-expansive.

Lemma 13. *The operators $\times$, $+$, ref, $\to$, cont, and comp shown in the lower part of Figure 4 are well-defined and non-expansive.*

For every $\Xi \vdash \tau$, define the non-expansive $[\![\tau]\!]_\Xi : \mathcal{T}^\Xi \to \mathcal{T}$ by induction on τ:

$$
\begin{aligned}
[\![\alpha]\!]_\Xi\,\varphi &= \varphi(\alpha)\\
[\![\mathsf{int}]\!]_\Xi\,\varphi &= \lambda\Delta.\,\{\,(in_{\mathbb{Z}}\,m, in_{\mathbb{Z}}\,m) \mid m \in \mathbb{Z}\,\}\\
[\![1]\!]_\Xi\,\varphi &= \lambda\Delta.\,\{\,(in_1\,*, in_1\,*)\,\}\\
[\![\tau_1 \times \tau_2]\!]_\Xi\,\varphi &= [\![\tau_1]\!]_\Xi\,\varphi \times [\![\tau_2]\!]_\Xi\,\varphi\\
[\![\tau_1 + \tau_2]\!]_\Xi\,\varphi &= [\![\tau_1]\!]_\Xi\,\varphi + [\![\tau_2]\!]_\Xi\,\varphi\\
[\![\mathsf{ref}\,\tau]\!]_\Xi\,\varphi &= \mathit{ref}([\![\tau]\!]_\Xi\,\varphi)\\
[\![\forall\alpha.\tau]\!]_\Xi\,\varphi &= \lambda\Delta.\,\{\,(in_\forall\,c, in_\forall\,c') \mid \forall\nu \in \mathcal{T}.\,(c,c') \in \mathit{comp}([\![\tau]\!]_{\Xi,\alpha}\,\varphi[\alpha \mapsto \nu])(\Delta)\,\}\\
[\![\mu\alpha.\tau]\!]_\Xi\,\varphi &= \mathit{fix}\left(\lambda\nu.\,\lambda\Delta.\,\{\,(in_\mu\,v, in_\mu\,v') \mid (v,v') \in [\![\tau]\!]_{\Xi,\alpha}\,\varphi[\alpha \mapsto \nu](\Delta)\,\}\right) \quad \text{(see Def. 14)}\\
[\![\tau_1 \to \tau_2]\!]_\Xi\,\varphi &= ([\![\tau_1]\!]_\Xi\,\varphi) \to (\mathit{comp}([\![\tau_2]\!]_\Xi\,\varphi))
\end{aligned}
$$

The following operators and elements are used above:

$$
\begin{aligned}
\times &: \mathcal{T} \times \mathcal{T} \to \mathcal{T} & \mathit{comp} &: \mathcal{T} \to \mathcal{T}_T\\
+ &: \mathcal{T} \times \mathcal{T} \to \mathcal{T} & \mathit{cont} &: \mathcal{T} \to \mathcal{T}_K\\
\mathit{ref} &: \mathcal{T} \to \mathcal{T} & \mathit{states} &\in \mathcal{T}_S\\
\to &: \mathcal{T} \times \mathcal{T}_T \to \mathcal{T} & R_{Ans} &\in \mathit{CRel}(Ans)
\end{aligned}
$$

$$
\begin{aligned}
(\nu_1 \times \nu_2)(\Delta) &= \{\,(in_\times(v_1,v_2), in_\times(v_1',v_2')) \mid (v_1,v_1') \in \nu_1(\Delta) \wedge (v_2,v_2') \in \nu_2(\Delta)\,\}\\
(\nu_1 + \nu_2)(\Delta) &= \{\,(in_+(\iota_1\,v_1), in_+(\iota_1\,v_1')) \mid (v_1,v_1') \in \nu_1(\Delta)\,\}\\
&\quad \cup \{\,(in_+(\iota_2\,v_2), in_+(\iota_2\,v_2')) \mid (v_2,v_2') \in \nu_2(\Delta)\,\}\\
\mathit{ref}(\nu)(\Delta) &= \{\,(\lambda_l,\lambda_l) \mid l \in \mathrm{dom}(\Delta) \wedge \forall\Delta_1 \geq \Delta.\ \mathit{App}\,(\Delta(l))\,(\Delta_1) = \nu(\Delta_1)\,\}\\
&\quad \cup \{\,(\lambda_l^{n+1},\lambda_l^{n+1}) \mid l \in \mathrm{dom}(\Delta) \wedge \forall\Delta_1 \geq \Delta.\ \mathit{App}\,(\Delta(l))\,(\Delta_1) =_n \nu(\Delta_1)\,\}\\
(\nu \to \xi)(\Delta) &= \{\,(in_\to f, in_\to f') \mid \forall\Delta_1 \geq \Delta.\forall(v,v') \in \nu(\Delta_1).(f\,v, f'\,v') \in \xi(\Delta_1)\,\}\\
\mathit{cont}(\nu)(\Delta) &= \{\,(k,k') \mid \forall\Delta_1 \geq \Delta.\forall(v,v') \in \nu(\Delta_1).\forall(s,s') \in \mathit{states}(\Delta_1).(k\,v\,s, k'\,v'\,s') \in R_{Ans}\,\}\\
\mathit{comp}(\nu)(\Delta) &= \{\,(c,c') \mid \forall\Delta_1 \geq \Delta.\forall(k,k') \in \mathit{cont}(\nu)(\Delta_1).\\
&\qquad \forall(s,s') \in \mathit{states}(\Delta_1).(c\,k\,s, c'\,k'\,s') \in R_{Ans}\,\}\\
\mathit{states}(\Delta) &= \{\,(s,s') \mid \mathrm{dom}(s) = \mathrm{dom}(s') = \mathrm{dom}(\Delta)\\
&\qquad \wedge\ \forall l \in \mathrm{dom}(\Delta).(s(l), s'(l)) \in \mathit{App}\,(\Delta(l))\,(\Delta)\,\}\\
R_{Ans} &= \{\,(\bot,\bot)\,\} \cup \{\,(\lfloor\iota_1\,m\rfloor, \lfloor\iota_1\,m\rfloor) \mid m \in \mathbb{Z}\,\}
\end{aligned}
$$

Fig. 4. Interpretation of types.

It is here, in order to show that *ref* is well-defined (and non-expansive), that we need the approximate locations λ_l^n. Suppose for the sake of argument that locations were modeled simply using a flat cpo of natural numbers, i.e., suppose that $Loc = \mathbb{N}_0$ and that $\pi_1(in_{Loc}\, l) = \lfloor in_{Loc}\, l \rfloor$ for all $l \in \mathbb{N}_0$. The definition of *ref* would then have the form $ref(\nu)(\Delta) = \{(in_{Loc}\, l, in_{Loc}\, l) \mid l \in \mathrm{dom}(\Delta) \wedge \dots\}$. The function $ref(\nu)$ from worlds to relations must be non-expansive. But assume then that $\Delta =_1 \Delta'$; then $ref(\nu)(\Delta) =_1 ref(\nu)(\Delta')$ by non-expansiveness, and hence $ref(\nu)(\Delta) = ref(\nu)(\Delta')$ since π_1 is the (lifted) identity on locations. In other words, $ref(\nu)$ would only depend on the "first approximation" of its argument world Δ: this can never be right, no matter what the particular definition of *ref* is.[6] This observation generalizes to variants where $\pi_n(in_{Loc}\, l) = \lfloor in_{Loc}\, l \rfloor)$ for some arbitrary finite n.

For any finite set Ξ of type variables, the set $\mathcal{T}^{\Xi}$ of functions from Ξ to $\mathcal{T}$ is a metric space with the product metric: $d'(\varphi, \varphi') = \max\{\, d(\varphi(\alpha), \varphi'(\alpha)) \mid \alpha \in \Xi \,\}$.

Definition 14. *Let τ be a type and let Ξ be a type environment such that $\Xi \vdash \tau$. The* relational interpretation of τ with respect to Ξ *is the non-expansive function $[\![\tau]\!]_{\Xi} : \mathcal{T}^{\Xi} \to \mathcal{T}$ defined by induction on τ in Figure 4. The interpretation of recursive types is by appeal to Banach's fixed-point theorem (see below).*

In more detail, the use of Banach's fixed point theorem in the interpretation of recursive types means that well-definedness of $[\![\tau]\!]_{\Xi}$ must be argued together with non-expansiveness, by induction on τ.[7] This is similar to the more familiar situation with the untyped semantics of terms presented in Section 3: there, well-definedness must be argued together with continuity because of the use of Kleene's fixed-point theorem in the interpretation of $\mathsf{fix}\, f.\lambda x.t$. The proof that $[\![\mu\alpha.\tau]\!]_{\Xi}$ is non-expansive uses Proposition 5.

We need the following substitution lemma, easily proved by induction on τ:

Lemma 15. *Let τ and τ' be types such that $\Xi, \alpha \vdash \tau$ and $\Xi \vdash \tau'$. For all $\varphi \in \mathcal{T}^{\Xi}$, $[\![\tau[\tau'/\alpha]]\!]_{\Xi}\, \varphi = [\![\tau]\!]_{\Xi,\alpha}\, (\varphi[\alpha \mapsto [\![\tau']\!]_{\Xi}\, \varphi])$.*

Corollary 16. $[\![\mu\alpha.\tau]\!]_{\Xi}\, \varphi = \lambda\Delta.\, \{\, (in_\mu\, v, in_\mu\, v') \mid (v, v') \in [\![\tau[\mu\alpha.\tau/\alpha]]\!]_{\Xi}\, \varphi\, \Delta \,\}$.

4.4 Interpretation of Terms

As for the interpretation of terms, we must show that the untyped meaning of a typed term is related to itself at the appropriate type. We first show that *comp* respects the operations of the monad T.

Definition 17. *For $\nu \in \mathcal{T}$ and $\xi \in \mathcal{T}_T$ and $\Delta \in \mathcal{W}$, let $\nu \overset{\Delta}{\to} \xi$ be the binary relation on functions $V \to TV$ defined by*

$$\nu \overset{\Delta}{\to} \xi = \{\, (f, f') \mid \forall \Delta_1 \geq \Delta. \forall (v, v') \in \nu(\Delta_1).\, (f\, v, f'\, v') \in \xi(\Delta_1) \,\}.$$

[6] In particular, the obvious definition of *ref* as $ref(\nu)(\Delta) = \{(in_{Loc}\, l, in_{Loc}\, l) \mid l \in \mathrm{dom}(\Delta) \wedge \forall \Delta_1 \geq \Delta.\ App\, (\Delta(l))\, (\Delta_1) = \nu(\Delta_1)\}$ would *not* be well-defined, since it would not be non-expansive in Δ.

[7] Non-expansiveness of $[\![\tau]\!]_{\Xi,\alpha}$ implies contractiveness of $\lambda\nu.\, \lambda\Delta.\, \{\, (in_\mu\, v, in_\mu\, v') \mid (v, v') \in [\![\tau]\!]_{\Xi,\alpha}\, \varphi[\alpha \mapsto \nu]\, (\Delta) \,\}$, as needed in the definition of $[\![\mu\alpha.\tau]\!]_{\Xi}\, \varphi$.

Proposition 18. *Let $\nu, \nu_1, \nu_2 \in \mathcal{T}$ and $\Delta \in \mathcal{W}$. (1) If $(v, v') \in \nu(\Delta)$, then $(\eta\, v, \eta\, v') \in comp(\nu)(\Delta)$. (2) If $(c, c') \in comp(\nu_1)(\Delta)$ and $(f, f') \in \nu_1 \xrightarrow{\Delta} comp(\nu_2)$, then $(c \star f,\, c' \star f') \in comp(\nu_2)(\Delta)$.*

Definition 19. *For every term environment $\Xi \vdash \Gamma$, every $\varphi \in \mathcal{T}^{\Xi}$, and every $\Delta \in \mathcal{W}$, let $[\![\Gamma]\!]_{\Xi}\, \varphi\, \Delta$ be the binary relation on $V^{\mathrm{dom}(\Gamma)}$ defined by*

$$[\![\Gamma]\!]_{\Xi}\, \varphi\, \Delta = \{\, (\rho, \rho') \mid \forall x \in \mathrm{dom}(\Gamma).\, (\rho(x), \rho'(x)) \in [\![\Gamma(x)]\!]_{\Xi}\, \varphi\, \Delta \,\}\,.$$

Definition 20. *Two typed terms $\Xi \mid \Gamma \vdash t : \tau$ and $\Xi \mid \Gamma \vdash t' : \tau$ of the same type are* semantically related, *written $\Xi \mid \Gamma \models t \sim t' : \tau$, if for all $\varphi \in \mathcal{T}^{\Xi}$, all $\Delta \in \mathcal{W}$, and all $(\rho, \rho') \in [\![\Gamma]\!]_{\Xi}\, \varphi\, \Delta$,*

$$\left([\![t]\!]_{\mathrm{dom}(\Gamma)}\, \rho, [\![t']\!]_{\mathrm{dom}(\Gamma)}\, \rho' \right) \in comp([\![\tau]\!]_{\Xi}\, \varphi)(\Delta)\,.$$

Theorem 21 (Fundamental Theorem). *Every typed term is semantically related to itself: if $\Xi \mid \Gamma \vdash t : \tau$, then $\Xi \mid \Gamma \models t \sim t : \tau$.*

Proof (sketch). By showing the stronger property that semantic relatedness is preserved by all the term constructs. The proof uses Proposition 18. □

Corollary 22 (Type soundness).

1. *If $\emptyset \mid \emptyset \vdash t : \tau$ is a closed term of type τ, then $[\![t]\!]_{\emptyset}\, \emptyset \neq error$.*
2. *If $\emptyset \mid \emptyset \vdash t : \mathsf{int}$ is a closed term of type int, then $[\![t]\!]^{\mathrm{p}} \neq error_{Ans}$.*

5 Examples

The model can be used to prove the equivalences in Section 5 of our earlier work [8]. More specifically, one can use the model to prove that some equivalences between different *functional* implementations of abstract data types are still valid in the presence of general references, and also prove some simple equivalences involving *imperative* abstract data types. (See Section 6 for more about extending the model to account properly for local state.) Here we only sketch one of these examples, as well as a "non-example": an equivalence that cannot be shown because of the existence of approximated locations in the model.

Example 23. We use the usual encoding of existential types by means of universal types [11]: $\exists\alpha.\tau = \forall\beta.(\forall\alpha.\, \tau \to \beta) \to \beta$. The type $\tau_{\mathrm{m}} = \exists\alpha.\, (1 \to \alpha) \times (\alpha \to 1) \times (\alpha \to \mathtt{int})$ can then be used to model imperative counter modules: the idea as that a value of type τ_{m} consists of some hidden type α, used to represent imperative counters, as well as three operations for creating a new counter, incrementing a counter, and reading the value of a counter, respectively.

Consider the following two module implementations, i.e., closed terms of type τ_{m}: $J = \Lambda\beta.\lambda c.\, c[\mathtt{ref\ int}]I$ and $J' = \Lambda\beta.\lambda c.\, c[\mathtt{ref\ int}]I'$ where

$$I = (\lambda x.\, \mathtt{ref}(0),\ \lambda x.\, x := !x + 1,\ \lambda x.\, !x)$$
$$I' = (\lambda x.\, \mathtt{ref}(0),\ \lambda x.\, x := !x - 1,\ \lambda x.\, -(!x))\,.$$

By parametricity reasoning, i.e., by exploiting the universal quantification in the interpretation of universal types, one can show that $\emptyset \mid \emptyset \models J \sim J' : \tau_{\mathrm{m}}$.

Example 24. Abbreviate $0 = \mu\alpha.\alpha$. One can show that 0 is an empty type: there are no closed values of type 0 and furthermore $[\![0]\!]_{\Xi}\,\varphi = \lambda\Delta.\emptyset$. Consider now the two terms $K = \lambda x.2$ and $K' = \lambda x.3$ of type $\mathsf{ref}\,0 \to \mathsf{int}$. Given a standard operational semantics for the language, a simple bisimulation-style argument should suffice to show that K and K' are contextually equivalent: no reference cell can ever contain a value of type 0, and therefore neither function can ever be applied. However, the equivalence $\emptyset \mid \emptyset \models K \sim K' : \mathsf{ref}\,0 \to \mathsf{int}$ does not hold. Briefly, the reason is the existence of approximated locations in the model.

6 Related Work

As already mentioned, the metric-space structure on uniform relations over universal domains is well-known [1, 4, 5, 10, 13]. The inverse-limit method for solving recursive domain equations was first adapted to metric spaces by America and Rutten [6]; see also Rutten [20]. For a unified account covering both domains and metric spaces, see Wagner [22]. Kripke logical relations are covered in Mitchell [14, Section 8.6] and in the references there.

Semantic (or "approximated") locations were first introduced in our earlier work [8]. That work contains an adequacy proof with respect to an operational semantics and an entirely different, quasi-syntactic interpretation of open types. Here we instead present an in some ways more natural interpretation that results from solving a recursive metric-space equation, thus obtaining a proper universe of semantic types. Open types are then interpreted in the expected way, i.e., as maps from environments of semantic types to semantic types.

The fundamental circularity between worlds and types in realizability-style possible-worlds models of polymorphism and general references was observed by Ahmed [2, p. 62] in the setting of operational semantics (and for unary relations). Rather than solve a recursive equation, her solution is to stratify worlds and types into different levels, represented by natural numbers. So-called step-indexing is used in the definition to ensure that a stratified variant of the fundamental theorem holds. These stratified worlds and types are somewhat analogous to the approximants of recursive-equation solutions that are employed in the inverse-limit method. The main advantage in "going to the limit" of the approximations and working with an actual solution (as we do here) is that approximation information is then not ubiquitous in definitions and proofs; by analogy, the only "approximation information" in our model is in the interpretation of references and in the requirement that user-supplied relations are uniform.[8]

Ahmed et al. [3] have recently (and independently) proposed a step-indexed model of a language very similar to ours, but in which worlds are defined in a more complicated way: this allows for proofs of much more advanced equivalences involving local state. We believe that our approach extends to this style of worlds and plan to investigate this further in future work: one potential advantage would be the removal of "approximation information" in definitions and equivalence

[8] In future work we plan to perform a more formal comparison.

proofs. We also plan to investigate local-state parameters [9]. In this article, we instead hope to have presented the fundamental ideas behind Kripke logical relations over recursively defined sets of worlds as needed for semantic modeling of parametric polymorphism, recursive types, and general references.

References

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

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

[3] A. Ahmed, D. Dreyer, and A. Rossberg. State-dependent representation independence. In *Proceedings of POPL*, 2009. To appear.

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

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

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

[7] N. Benton and B. Leperchey. Relational reasoning in a nominal semantics for storage. In *Proceedings of TLCA*, number 3461 in LNCS, pages 86–101, 2005.

[8] L. Birkedal, K. Støvring, and J. Thamsborg. Relational parametricity for references and recursive types. In *Proceedings of TLDI*, 2009. To appear.

[9] N. Bohr and L. Birkedal. Relational reasoning for recursive types and references. In *Proceedings of APLAS*, number 4279 in LNCS, pages 79–96, 2006.

[10] F. Cardone. Relational semantics for recursive types and bounded quantification. In *Proceedings of ICALP*, number 372 in LNCS, pages 164–178, 1989.

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

[12] P. B. Levy. Call-by-push-value: Decomposing call-by-value and call-by-name. *Higher-Order and Symbolic Computation*, 19(4):377–414, 2006.

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

[14] J. C. Mitchell. *Foundations for Programming Languages*. The MIT Press, 1996.

[15] E. Moggi. Notions of computation and monads. *Information and Computation*, 93:55–92, 1991.

[16] R. L. Petersen, L. Birkedal, A. Nanevski, and G. Morrisett. A realizability model for impredicative Hoare type theory. In *Proceedings of ESOP*, number 4960 in LNCS, pages 337–352, 2008.

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

[18] A. M. Pitts. Relational properties of domains. *Information and Computation*, 127:66–90, 1996.

[19] G. D. Plotkin and J. Power. Computational effects and operations: An overview. *Electronic Notes in Theoretical Computer Science*, 73:149–163, 2004.

[20] J. J. M. M. Rutten. Elements of generalized ultrametric domain theory. *Theoretical Computer Science*, 170(1-2):349–381, 1996.

[21] M. B. Smyth and G. D. Plotkin. The category-theoretic solution of recursive domain equations. *SIAM Journal on Computing*, 11(4):761–783, 1982.

[22] K. R. Wagner. *Solving Recursive Domain Equations with Enriched Categories*. PhD thesis, Carnegie Mellon University, 1994.