First steps in synthetic guarded domain theory: step-indexing in the topos of trees

Lars Birkedal
IT University of Copenhagen
Rasmus Ejlers Møgelberg
IT University of Copenhagen
Jan Schwinghammer
Saarland University
Kristian Støvring
DIKU, University of Copenhagen

Abstract

We present the topos $S$ of trees as a model of guarded recursion. We study the internal dependently-typed higher-order logic of $S$ and show that $S$ models two modal operators, on predicates and types, which serve as guards in recursive definitions of terms, predicates, and types. In particular, we show how to solve recursive type equations involving dependent types. We propose that the internal logic of $S$ provides the right setting for the synthetic construction of abstract versions of step-indexed models of programming languages and program logics. As an example, we show how to construct a model of a programming language with higher-order store and recursive types entirely inside the internal logic of $S$.

1. Introduction

Recursive definitions are ubiquitous in computer science. In particular, in semantics of programming languages and program logics we often use recursively defined functions and relations, and also recursively defined types (domains). For example, in recent years there has been extensive work on giving semantics of type systems for programming languages with dynamically allocated higher-order store, such as general ML-like references. Models have been expressed as Kripke models over a recursively defined set of worlds (an example of a recursively defined domain) and have involved recursively defined relations to interpret the recursive types of the programming language; see [4] and the references therein.

In this paper we study a topos $S$, which we show models guarded recursion in the sense that it allows for guarded recursive definitions of both recursive functions and relations as well as recursive types. The topos $S$ is known as the topos of trees (or forests); what is new here is our application of this topos to model guarded recursion.

The internal logic of $S$ is a standard many-sorted higher-order logic extended with modal operators on both types and terms. (Recall that terms in higher-order logic include both functions and relations, as the latter are simply Prop-valued functions.) This internal logic can then be used as a language to describe semantic models of programming languages with the features mentioned above. As an example which uses both recursively defined types and recursively defined relations in the $S$-logic, we present a model of $F_{\mu,\text{ref}}$, a call-by-value programming language with impredicative polymorphism, recursive types, and general ML-like references.

To situate our work in relation to earlier work, we now give a quick overview of the technical development of the present paper followed by a comparison to related work. We end the introduction with a summary of our contributions.

Overview of technical development. The topos $S$ is the category of presheaves on $\omega$, the first infinite ordinal. This topos is known as the topos of trees, and is one of the most basic examples of presheaf categories.

There are several ways to think intuitively about this topos. Let us recall one intuitive description, which can serve to understand why it models guarded recursion. An object $X$ of $S$ is a contravariant functor from $\omega$ (viewed as a preorder) to $\text{Set}$. The internal logic of $S$ is an extension of standard Kripke semantics: for constant sets, the truth value of a predicate is just the set of worlds (downwards closed subsets of $\omega$) for which the predicate holds. This observation suggests that
there is a modal “later” operator $\triangleright$ on predicates $\Omega^{\Delta(S)}$ on constant sets, similar to what has been studied earlier [3, 9].

Intuitively, for a predicate $\varphi : \Omega^{\Delta(S)}$ on constant set $\Delta(S)$, $\triangleright(\varphi)$ contains $n + 1$ if $\varphi$ contains $n$. (A future world is a smaller number, hence the name “later” for this operator.)

A recursively specified predicate $\mu r.\varphi(r)$ is well-defined if every occurrence of the recursion variable $r$ in $\varphi$ is guarded by a $\triangleright$ modality: by definition of $\triangleright$, to know whether $n + 1$ is in the predicate it suffices to know whether $n$ is in the predicate. There is also an associated L"ob rule for induction, $(\triangleright \varphi \rightarrow \varphi) \rightarrow \varphi$, as in [3].

Here we show that in fact there is a later operator not only on predicates on constant sets, but also on predicates on general variable sets, with associated L"ob rule, and well-defined guarded recursive definitions of predicates.

Moreover, there is also a later operator $\triangleright$ (a functor) on the variable sets themselves: $\triangleright(X)$ is given by $\triangleright(X)(1) = \{ \ast \}$ and $\triangleright(X)(n + 1) = X(n)$. We can show the well-definedness of recursive variable sets $\mu X.F(X)$ in which the recursion variable $X$ is guarded by this operator $\triangleright$. Intuitively, such a recursively specified variable set is well-defined since by definition of $\triangleright$, to know what $\mu X.F(X)$ is at level $n + 1$ it suffices to know what it is at level $n$.

In the technical sections of the paper, we make the above precise. In particular, we detail the internal logic and the use of later on functions / predicates and on types. We explain how one can solve mixed-variance recursive type equations, for a wide collection of types. We show how to use the internal logic of $\mathcal{S}$ to give a model of $F_{\mu,ref}$. The model, including the operational semantics of the programming language, is defined completely inside the internal logic; we discuss the connection between the resulting model and earlier models by relating internal definitions in the internal logic to standard (external) definitions. Since $\mathcal{S}$ is a topos, $\mathcal{S}$ also models dependent types. We give technical semantic results as needed for using later on dependent types and for recursive type-equations involving dependent types. We think of this as a first step towards a formalized dependent type theory with a later operator; here we focus on the foundational semantic issues.

To explain the relationship to some of the related work, we point out that $\mathcal{S}$ is equivalent to the category of sheaves on $\mathcal{S}$, where $\mathcal{S}$ is the complete Heyting algebra of natural numbers with the usual ordering and extended with a top element $\infty$. Moreover, this sheaf category, and hence also $\mathcal{S}$, is equivalent to the topos obtained by the tripos-to-topos construction [19] applied to the tripos $\mathsf{Set}(\_,$ $\mathcal{S})$. The logic of constant sets in $\mathcal{S}$ is exactly the logic of this tripos.$^1$

In this paper we work with the presentation of $\mathcal{S}$ as presheaves since it is the most concrete, but many of our results generalize to sheaf categories over other complete well-founded Heyting algebras.

Related work. Nakano presented a simple type theory with a guarded recursive types [24] which can be modelled using complete bounded ultrametric spaces [5]. We show in Section 5 that the category $\mathsf{BiCBUlt}$ of bisected, complete bounded ultrametric spaces is a co-reflective subcategory of $\mathcal{S}$. Thus, our present work can be seen as an extension of the work of Nakano to include the full internal language of a topos, in particular dependent types, and an associated higher-order logic. Pottier [26] presents an extension of System F with recursive kinds based on Nakano’s calculus; hence $\mathcal{S}$ also models the kind language of his system.

Di Gianantonio and Miculan [8] studied guarded recursive definitions of functions in certain sheaf toposes over well-founded complete Heyting algebras, thus including $\mathcal{S}$. Our work extends the work of Di Gianantonio and Miculan by also including guarded recursive definitions of types, by emphasizing the use of the internal logic (this was suggested as future work in [8]), and by including an extensive example application.

In our earlier work, we advocated the use of complete bounded ultrametric spaces for solving recursive type and relation equations that come up when modelling programming languages with higher-order store [4, 6]. As mentioned above, $\mathsf{BiCBUlt}$ is a subcategory of $\mathcal{S}$, and thence our present work can be seen as an improvement of this earlier work: it is an improvement since $\mathcal{S}$ supports full higher-order logic. In the earlier work, we had to show that the functions we defined in the interpretation of the programming language types were non-expansive. Here we take the synthetic approach (cf. [18]) and place ourselves in the internal logic of the topos when defining the interpretation of the programming language, see Section 3. This means that there is no need to prove properties like non-expansiveness since, intuitively, all functions in the topos are suitably non-expansive.

Dreyer et al. [9] proposed a logic, called LSLR, for defining step-indexed interpretations of programming languages with recursive types, building on earlier work by Appel et al. [3] who proposed the use of a later modality on predicates. The point of LSLR is that it provides for more abstract ways of constructing and reasoning with step-indexed models, thus avoiding tedious calculations with step indices. The core logic of LSLR is the logic of the tripos $\mathsf{Set}(\_,$ $\mathcal{S})$ mentioned above,$^2$ which allows for recursively defined predicates following [3], but not recursively defined types. One point of passing from this tripos to the topos $\mathcal{S}$ is that it gives us a wider collection of types.

---

$^1$Recall that the tripos $\mathsf{Set}(\_,$ $\mathcal{S})$ is a model of logic in which types and terms are interpreted as sets and functions, and predicates are interpreted as $\mathcal{S}$-valued functions.

$^2$Dreyer et al. [9] presented the semantics of their second-order logic in more concrete terms, avoiding the use of triposes, but it is indeed a fragment of the internal logic of the mentioned tripos.
Dreyer et al. developed an extension of LSLR called LADR for reasoning about step-indexed models of the programming language $F_{\mu, \text{ref}}$ with higher-order store [11]. LADR is a specialized logic where much of the world structure used for reasoning efficiently about local state is hidden by the model of the logic; here we are proposing a general logic that can be used to construct many step-indexed models, including the one used to model LADR. In particular, in our example application in Section 3, we define a set of worlds inside the $S$ logic, using recursively defined types.

As part of our analysis of recursive dependent types, we define a class of types, called functorial types, by a grammar and some simple logical conditions. We show that functorial types are closed under nested recursive types, a result which is akin to results on nested inductive types [1, 12].

The natural numbers object $N$ in $S$ is the constant set of natural numbers.

2. The $S$ Topos

The category $S$ is that of presheaves on $\omega$, the preorder of natural numbers starting from 1 and ordered by inclusion. Explicitly, the objects of $S = \text{Set}^{\omega^{op}}$ are families of sets indexed by natural numbers together with restriction maps $r_n: X(n + 1) \to X(n)$. Morphisms are families $(f_n)_n$ of maps commuting with the restriction maps as indicated in

\[ X(1) \leftarrow X(2) \leftarrow X(3) \leftarrow \ldots \]

\[ f_1 \quad f_2 \quad f_3 \]

\[ Y(1) \leftarrow Y(2) \leftarrow Y(3) \leftarrow \ldots \]

If $x \in X(m)$ and $n \leq m$ we write $x|_n$ for $r_n \circ \cdots \circ r_{m-1}(x)$.

As all presheaf categories, $S$ is a topos, i.e., it is cartesian closed and has a subobject classifier. Moreover, it is complete and cocomplete, and limits and colimits are computed pointwise. The $n$th component of the exponent $X^N(n)$ is the set of tuples $(f_1, \ldots, f_n)$ commuting with the restriction maps, and the restriction maps of $X^N$ are given by projection.

A subobject $A$ of $X$ is a family of subsets $A(n) \subseteq X(n)$ such that $r_n(A(n + 1)) \subseteq A(n)$. The subobject classifier has $\Omega(n) = \{0, \ldots, n\}$ and restriction maps $r_n(x) = \min(n, x)$. The characteristic morphism $\chi_A: X \to \Omega$ maps $x \in X(n)$ to the maximal $m$ such that $x|_m \in A(m)$ if such an $m$ exists and 0 otherwise.

The natural numbers object $N$ in $S$ is the constant set of natural numbers.

The endofunctor. Define the functor $\triangleright: S \to S$ by $\triangleright X(1) = \{\ast\}$ and $\triangleright X(n + 1) = X(n)$. This functor, called later, has a left adjoint (so $\triangleright$ preserves all limits) given by $\triangleright X(n) = X(n + 1)$. Since limits are computed pointwise, $\triangleright$ preserves them, and so the adjunction $\triangleright \dashv \triangleright$ defines a geometric morphism, in fact an embedding. However, we shall not make use of this fact in the present paper (because $\triangleright$ is not a fibred endo-functor on the codomain fibration, hence is not a useful operator in the dependent type theory; see Section 4).

There is a natural transformation $\text{next}_X: X \to \triangleright X$ whose 1st component is the unique map into $\{\ast\}$ and whose $(n + 1)$st component is $r_n$. Since $\triangleright$ preserves finite limits, there is always a morphism

\[ J: \triangleright(X \to Y) \to (\triangleright X \to \triangleright Y). \]  

An operator on predicates. There is a morphism $\triangleright: \Omega \to \Omega$ mapping $n \in \Omega(m)$ to $\min(m, n + 1)$. By setting $\chi_{\triangleright A} = \triangleright \circ \chi_A$ there is an induced operation on subobjects, again denoted $\triangleright$. This operation, which we also call later, is connected to the $\triangleright$ functor, since there is a pullback diagram

$\triangleright m \quad \triangleright A$

$\text{next}_X \quad \triangleright m$

for any subobject $m: A \to X$.
Recursive morphisms. We introduce a notion of contractive morphism and show that these have unique fixed points.

**Definition 2.1.** A morphism $f : X \to Y$ is contractive if there exists a morphism $g : \Box X \to Y$ such that $f = g \circ \text{next}_X$. A morphism $f : X \times Y \to Z$ is contractive in the first variable if there exists $g$ such that $f = g \circ \text{next}_X \times \text{id}_Y$.

For instance, contractiveness of $\triangleright$ on $\Omega$ is witnessed by $\text{succ} : \Box \Omega \to \Omega$ with $\text{succ}_n(k) = k + 1$.

If $f : X \to Y$ is contractive then the value of $f_{n+1}(x)$ can be computed from $r_n(x)$ and moreover, $f_1$ must be constant. If $X = Y$ we can define a fixed point $x : 1 \to X$ by defining $x_1 = g_1(\ast)$ and $x_{n+1} = g_{n+1}(x_n)$. This construction can be generalized to include fixed points of morphisms with parameters as follows.

**Theorem 2.2.** There exists a natural family of morphisms $\text{fix}_X : (\Box X) \to X$, indexed by the collection of all objects $X$, which computes unique fixed points in the sense that if $f : X \times Y \to X$ is contractive in the first variable as witnessed by $g$, i.e., $f = g \circ \text{next}_X \times \text{id}_Y$, then $\text{fix}_X \circ g$ is the unique $h : Y \to X$ such that $f \circ \langle h, \text{id}_Y \rangle = h$.

### 2.1 Internal logic

We start by calling to mind parts of the Kripke-Joyal forcing semantics for $S$. For $X \in S$, $\varphi : X_1 \times \cdots \times X_m \to \Omega$, $n \in \omega$, and $\alpha_1 \in X_1(n), \ldots, \alpha_m \in X_m(n)$, we define $n \models \varphi(\alpha_1, \ldots, \alpha_m)$ iff $\varphi(n)(\alpha_1, \ldots, \alpha_m) = n$.

The standard clauses for the forcing relation are as follows [21, 22] (we write $\overline{\alpha}$ for a sequence $\alpha_1, \ldots, \alpha_m$):

- $n \models (s = t)_{\overline{\alpha}}$ if and only if $[s]_{\overline{\alpha}} = [t]_{\overline{\alpha}}$.
- $n \models R(t_1, \ldots, t_k)_{\overline{\alpha}}$ if and only if $[R]_n ([t_1]_{\overline{\alpha}}, \ldots, [t_k]_{\overline{\alpha}}) = 1$.
- $n \models (\varphi \land \psi)_{\overline{\alpha}}$ if and only if $n \models \varphi_{\overline{\alpha}} \land n \models \psi_{\overline{\alpha}}$.
- $n \models (\varphi \lor \psi)_{\overline{\alpha}}$ if and only if $n \models \varphi_{\overline{\alpha}} \lor n \models \psi_{\overline{\alpha}}$.
- $n \models (\varphi \circ \psi)_{\overline{\alpha}}$ if and only if $\forall k \leq n. \forall \alpha \in [K](k). n \models \varphi(\overline{\alpha}, \alpha)$.
- $n \models (\forall x : X. \varphi(x))_{\overline{\alpha}}$ if and only if $\forall \alpha \in [X](n). n \models \varphi(\overline{\alpha}, \alpha)$.
- $n \models (\exists x : X. \varphi(x))_{\overline{\alpha}}$ if and only if $\exists \alpha \in [X](n). n \models \varphi(\overline{\alpha}, \alpha)$.

### Proposition 2.3.

$\triangleright$ is the unique morphism on $\Omega$ satisfying $1 \models \triangleright \varphi(\alpha)$ and $n + 1 \models \triangleright \varphi(\alpha)$ iff $n \models \varphi(\alpha|_n)$. Moreover, $\forall x, y : X. \triangleright (x = y) \iff \text{next}_X(x) = \text{next}_X(y)$ holds in $S$.

The following definition will be useful for presenting facts about the internal logic of $S$.

**Definition 2.4.** An object $X$ in $S$ is total if all the restriction maps $r_n$ are surjective.

Hence all constant objects $\Delta(S)$ are total, but the total objects also include many non-constant objects, e.g., the subobject classifier. The above definition is phrased in terms of the model; the internal logic can be used to give a simple characterization of when $X$ is total and inhabited by a global element: that is the case iff $\text{next}_X$ is internally surjective in $S$, i.e., $\forall y : \Box X. \exists x : X. \text{next}_X(x) = y$ holds in $S$. The following proposition can be proved using the forcing semantics; note that the distribution rules below for $\triangleright$ generalize the ones for constant sets described in [9] (since constant sets are total).

**Proposition 2.5.** In the internal logic of $S$ we have:

1. (Monotonicity). $\forall p : \Omega. p \to \triangleright p$.
2. (Löb rule). $\forall p : \Omega. (\triangleright p \to p) \to \triangleright p$.
3. $\triangleright$ commutes with the logical connectives $\top, \land, \to, \lor$, but does not preserve $\bot$.
4. For all $X, Y$, and $\varphi$, we have $\exists y : Y. \triangleright \varphi(x, y) \to \triangleright (\exists y : Y. \varphi(x, y))$. The implication in the opposite direction holds if $Y$ is total and inhabited.
5. For all $X, Y$, and $\varphi$, we have $\triangleright (\forall y : Y. \varphi(x, y)) \to \forall y : Y. \triangleright \varphi(x, y)$. The implication in the opposite direction holds if $Y$ is total.

We now define an internal notion of contractiveness in the logic of $S$ which implies (in the logic) the existence of a unique fixed point for inhabited types.

**Definition 2.6.** The predicate $\text{Con}$ on $Y^X$ is defined in the internal logic by

$\text{Con}(f) \overset{\text{def}}{=} \forall x, x' : X. \triangleright (x = x') \to f(x) = f(x')$.

This notion of contractiveness generalizes the usual metric notion of contractiveness for functions between complete bounded ultrametric spaces; see Section 5.

**Theorem 2.7.** (Internal Banach Fixed-Point Theorem). The following holds in $S$:

$\exists x : X. \top \land \text{Con}(f) \to \exists ! x : X. f(x) = x$.

For a morphism $f : X \to Y$, corresponding to a global element of $Y^X$, we have that if $f$ is contractive (in the external sense of Definition 2.1), then $\text{Con}(f)$ holds in the logic of $S$. The converse is true if $X$ is total and inhabited, but not in general. We use both notions of contractiveness: the external notion provides for a simple algebraic theory of fixed points for not only morphisms but also functions (see Section 2.2), whereas the internal notion is useful when working in the internal logic.

The above theorem (the Internal Banach Fixed-Point Theorem) is proved in the internal logic using the following lemma, which expresses a non-classical property. The lemma can be proved using the Löb rule (and using that $N$ is a total object).
Lemma 2.8. The following holds in $S$:

$$\text{Contr}(f) \rightarrow \exists n : N, \forall x, x' : X, f^n(x) = f^n(x').$$

Recursive relations. As an example application of Theorem 2.7, we consider the definition of recursive predicates. Let $\varphi(r) : \Omega^X$ be a predicate on $X$ in the internal logic of $S$ as presented above (over non-dependent types, but possibly using $\triangleright$) with free variable $r$, also of type $\Omega^X$. Note that $\Omega^X$ is inhabited by a global element. If $r$ only occurs under a $\triangleright$ in $\varphi$, then $\varphi$ defines an internally contractive map $\varphi : \Omega^X \rightarrow \Omega^X$ (proved by external induction on $\varphi$). Therefore, by Theorem 2.7, $\exists r : \Omega^X, \varphi(r) = r$ holds in $S$. By description (aka axiom of unique choice), which holds in any topos [21], there is then a morphism $R : 1 \rightarrow \Omega^X$ such that $\varphi(R) = R$ in $S$, and since internal and external equality coincides, also $\varphi(R) = R$ externally as morphisms $1 \rightarrow \Omega^X$. In summa, we have shown the well-definedness of recursive predicates $r = \varphi(r)$ where $r$ only occurs guarded by $\triangleright$ in $\varphi$.

Note that we have proved the existence of recursive guarded relations (and thus do not have to add them to the language with special syntax) since we are working with a higher-order logic.

Example 2.9. Suppose $R \subseteq X \times X$ is some relation on a set $X$. We can include it into $S$ by using the functor $\Delta : \text{Set} \rightarrow S$, obtaining $\Delta R \subseteq \Delta X \times \Delta X$. Consider the recursive relation

$$R^e(x, y) \overset{\text{def}}{\iff} (x = y) \lor \exists z : (\Delta R(x, z) \wedge \triangleright R^e(z, y)).$$

Now, $1 \vdash R^e(x, y)$ always holds and $n + 1 \vdash R^e(x, y)$ iff $(x, y) \in \cup_{0 \leq i \leq n} R^i$ or there exists $z$ such that $R^e(x, z)$ holds. If $R$ is a rewrite relation then $n + 1 \vdash R^e(x, y)$ states the extent to which we can determine if $x$ rewrites to $y$ by inspecting all rewrite sequences of length at most $n - 1$.

A variant of Example 2.9 is used in Section 3.

2.2 Recursive domain equations

In this section we present a simplified version of our results on solutions to recursive domain equations in $S$ sufficient for the example of Section 3. The full results on recursive domain equations can be found in Section 4.

Denote by $\triangleright f^n : 1 \rightarrow Y^X$ the curried version of $f : X \rightarrow Y$. Following Kock [20] we say that an endofunctor $F : S \rightarrow S$ is strong if, for all $X, Y$, there exists a morphism $F_{X,Y} : Y^X \rightarrow FY^FX$ such that $F_{X,Y} \circ \triangleright f^n = \triangleright Ff^n$ for all $f$.

Definition 2.10. A strong endofunctor on $S$ is locally contractive if each $F_{X,Y}$ is contractive.

This notion readily generalizes to mixed-variance endofunctors on $S$. For example, $\triangleright$ is locally contractive, and one can show that the composition of a strong functor and a locally contractive functor (in either order) is locally contractive. As a result, one can show that any type expression $A(X, Y)$ constructed from type variables $X, Y$ using $\triangleright$ and simple type constructors in which $X$ occurs only negatively and $Y$ only positively and both only under $\triangleright$ gives rise to a locally contractive functor.

Theorem 2.11. Let $F : S^{op} \times S \rightarrow S$ be a locally contractive functor. Then there exists a unique $X$ (up to isomorphism) such that $F(X, X) \cong X$.

Although there is no space for a full proof of Theorem 2.11 we sketch the construction to illustrate the use of the locally contractive functors. We consider first the covariant case.

Lemma 2.12. Let $F : S \rightarrow S$ be locally contractive and say that $f : X \rightarrow Y, g : Y \rightarrow X$ is an $n$-isomorphism pair if $f_i$ is inverse to $g_i$ for all $i \leq n$. Then $F$ maps $n$-isomorphism pairs to $n + 1$-isomorphism pairs for all $n$.

Construct morphisms $p = F! : F^2 \rightarrow F$ and $e$ as the composition

$$F1 \xrightarrow{\delta} F1 \times F1 \xrightarrow{\text{st}} 1 \times F^2 \cong F^2,$$

where $\delta$ is the diagonal and $\text{st}$ is the strength corresponding to $F_{\omega}$ [20]. By Lemma 2.12 $(F^n p, F^n e)$ is an $n$-isomorphism pair, and so intuitively one can construct a fixed point for $F$ by taking the $n$th component to be $F^{n+1}(n)$. For our formal proof we derived a limit / colimit coincidence of the sequence of injection / projection pairs

$$F1 \xleftarrow{p} F^2 \xleftarrow{Fp} F^3 \xleftarrow{F^2 p} F^4 \ldots$$

Any fixed point for such an $F$ must be at the same time an initial algebra and a final coalgebra: given any fixed point $f : FX \cong X$ and algebra $g : FY \rightarrow Y$ a morphism $h : X \rightarrow Y$ is a homomorphism iff $\triangleright h \triangleright = h \circ Fk \circ f^{-1}$. Since $F$ is locally contractive, $\xi$ is contractive and so must have a unique fixed point. The case of final coalgebras is similar.

Thus, $S$ is algebraically compact in the sense of Freyd [13–15] with respect to locally contractive functors. The solutions to general recursive domain equations can then be established using Freyd’s constructions.

3. Application to Step-Indexing

As an example, we now construct a model of a programming language with higher-order store and recursive types.
entirely inside the internal logic of \( S \). There are two points we wish to make here. First, although the programming language is quite expressive, the internal model looks—almost—like a naive, set-theoretic model. The exception is that guarded recursion is used in a few, select places, such as defining the meaning of recursive types, where the naive approach would fail. Second, when viewed externally, we recover a standard, step-indexed model. This example therefore illustrates that the topos of trees gives rise to simple, synthetic accounts of step-indexed models.

All definitions and results in Sections 3.1 to 3.4 are in the internal logic of \( S \). In Section 3.5 we investigate what these results mean externally.

### 3.1 Language

The types and terms of \( F_{\mu, \text{ref}} \) are as follows:

\[
\begin{align*}
\tau &::= 1 \mid \tau_1 \times \tau_2 \mid \mu \alpha. \tau \mid \forall \alpha. \tau \mid \alpha \mid \tau_1 \rightarrow \tau_2 \mid \text{ref} \tau \\
t &::= x \mid l \mid () \mid (t_1, t_2) \mid \text{fst} t \mid \text{snd} t \mid \text{fold} t \mid \text{unfold} t \\
\Lambda \alpha. t &\mid t[\tau] \mid \Lambda x. t \mid t_1 t_2 \mid \text{ref} t \mid !t \mid t_1 := t_2
\end{align*}
\]

(The full term language also includes sum types, and can be found in Appendix A.) Here \( l \) ranges over location constants, which are encoded as natural numbers.

More explicitly, the sets \( \text{OType} \) and \( \text{OTerm} \) of possibly open types and terms are defined by induction according to the grammars above (using that toposes model \( W \)-types [23]), and then by quotienting with respect to \( \alpha \)-equivalence.

The set \( \text{OValue} \) of syntactic values is an inductively defined subset of \( \text{OTerm} \):

\[
v ::= x \mid l \mid () \mid (v_1, v_2) \mid \text{fold} v \mid \Lambda \alpha. t \mid \Lambda x. t
\]

Let \( \text{Term} \) and \( \text{Value} \) be the subsets of closed terms and closed values, respectively. Let \( \text{Store} \) be the set of finite maps from natural numbers to closed values; this is encoded as the set of those finite lists of pairs of natural numbers and closed values that contain no number twice. Finally, let \( \text{Config} = \text{Term} \times \text{Store} \).

The typing judgements have the form \( \Xi \mid \Gamma \vdash t : \tau \) where \( \Xi \) is a context of type variables and \( \Gamma \) is a context of term variables. The typing rules are standard and can be found in Appendix A. Notice, however, that there is no context of location variables and no typing judgement for stores: we only need to type-check terms that can occur in programs.

### 3.2 Operational semantics

We define a standard one-step relation \( \text{step} : \mathcal{P}(\text{Config} \times \text{Config}) \) on configurations by induction, following the usual presentation of such relations by means of inference rules. For simplicity, allocation is deterministic: when allocating a new reference cell, we choose the smallest location not already in the store. Notice that the \( \text{step} \) relation is defined on untyped configurations. Erroneous configurations are “stuck.”

So far, we have defined the language and operational semantics exactly as we would in standard set theory. Next comes the crucial difference. We use Theorem 2.7 to define the predicate \( \text{eval} : \mathcal{P}(\text{Term} \times \text{Store} \times \mathcal{P}(\text{Value} \times \text{Store})) \),

\[
\text{eval}(t,s,Q) \defeq (t \in \text{Value} \land Q(t,s)) \lor (\exists t_1 : \text{Term}, s_1 : \text{Store}. \text{step}((t,s), (t_1, s_1)) \land \triangleright \text{eval}(t_1, s_1, Q))
\]

Intuitively, the predicate \( Q \) is a post-condition, and \( \text{eval}(t,s,Q) \) is a partial correctness specification, in the sense of Hoare logic, meaning the following: (1) The configuration \( (t,s) \) is safe, i.e., it does not lead to an error. (2) If the configuration \( (t,s) \) evaluates to some pair \((v,s)\), then at that point in time \((v,s)\) satisfies \( Q \). We shall justify this intuition in Section 3.5 below. The use of \( \triangleright \) ensures that the predicate is well-defined; in effect, we postulate that one evaluation step in the programming language actually takes one unit of time in the sense of the internal logic. As we shall see below, this “temporal” semantics is essential in the proof of the fundamental theorem of logical relations.

Notice how guarded recursion is used to give a simple, coinduction-style definition of partial correctness. The L"ob rule can then be conveniently used for reasoning about this definition. For example, the rule gives a very easy proof that if \( (t,s) \) is a configuration that reduces to itself in the sense that \( \text{step}((t,s), (t,s)) \) holds, then \( \text{eval}(t,s,Q) \) holds for any \( Q \). The L"ob rule also proves the following results, which are used to show the fundamental theorem below.

**Proposition 3.1.** Let \( Q, Q' \in \mathcal{P}(\text{Value} \times \text{Store}) \) such that \( Q \subseteq Q' \). Then for all \( t \) and \( s \) we have that \( \text{eval}(t,s,Q) \) implies \( \text{eval}(t,s,Q') \).

**Proposition 3.2.** For all stores \( s \), all terms \( t \), all evaluation contexts \( E \) such that \( E[t] \) is closed, and all predicates \( Q \in \mathcal{P}(\text{Value} \times \text{Store}) \), we have that \( \text{eval}(E[t],s,Q) \) holds iff \( \text{eval}(t,s, \lambda (v_1,s_1). \text{eval}(E[v_1], s_1, Q)) \) holds.

### 3.3 Definition of Kripke worlds

The main idea behind our interpretation of types is as in [4, 7]: Since \( F_{\mu, \text{ref}} \) includes reference types, we use a Kripke model of types, where a semantic type is defined to be a world-indexed family of sets of syntactic values. A world is a map from locations to semantic types. This introduces a circularity between semantic types \( \mathcal{T} \) and worlds \( \mathcal{W} \).
We solve the circularity using guarded recursion. More precisely, we define the set
\[ \tilde{T} = \mu X. \bigtriangledown(N \rightarrow_{\text{fin}} X) \rightarrow_{\text{mon}} P(\text{Value}) \]  
Here \( N \rightarrow_{\text{fin}} X \) is the set \( \sum_{A : P_{\text{fin}}(N)} X^A \) where \( P_{\text{fin}}(N) = \{ A \subseteq N \mid \exists m \forall n \in A. n < m \} \) ordered by graph inclusion and \( \rightarrow_{\text{mon}} \) is the set of monotonic functions realized as a subset type over the function space.

One way to see that \( \tilde{T} \) is well-defined is to check that it is a functorial type in the sense of Section 4. Alternatively, observe that the corresponding functor is of the form \( F = \bigtriangledown \circ G \). Here \( G \) is strong because its action on morphisms can be defined as a term \( Y^X \to GY^G X \) in the internal logic. Now, since \( \bigtriangledown \) is locally contractive so is \( F \).

Hence by Theorem 2.11, \( F \) has a unique fixed point \( \check{T} \), with an isomorphism \( i : \check{T} \to F(\check{T}) \). We define
\[ W = N \rightarrow_{\text{fin}} \check{T}, \quad T = W \rightarrow_{\text{mon}} P(\text{Value}) \]  
and \( T^* = W \rightarrow P(\text{Term}) \). Notice that \( \check{T} \) is isomorphic to \( \bigtriangledown T \). We now define \( \text{app} : \check{T} \rightarrow T \) and \( \text{lam} : T \rightarrow \check{T} \) as follows. First, \( \text{app} \) is the isomorphism \( i \) composed with the operator \( d : \bigtriangledown T \rightarrow T \) given by
\[ d(f) = \lambda w. \lambda v. \text{succ}(J(f)(\text{nextw}))(\text{nextv}), \]  
where \( J \) is the map in (1) and \( \text{succ} : \bigtriangledown \Omega \rightarrow \Omega \) is as defined on page 4. (This is a general way of lifting algebras for \( \bigtriangledown \) to function spaces.) Here one needs to check that \( d \) is well-defined, i.e., preserves monotonicity. Second, \( \text{lam} : T \rightarrow \check{T} \) is defined by \( \text{lam} = i^{-1} \circ \text{next} \).

Define \( \triangleright : \Omega \rightarrow \check{T} \) as the pointwise extension of \( \triangleright : \Omega \rightarrow \Omega \), i.e., for \( \nu \in T, w \in W \) and \( v \in \text{Value} \), we have that \( \triangleright(v)(w)(v) \) holds iff \( \triangleright(v(w))(v) \) holds.

**Lemma 3.3.** \( \text{app} \circ \text{lam} = \triangleright : T \rightarrow \check{T} \).

### 3.4 Interpretation of types

Let \( \text{TVar} \) be the set of type variables, and for \( \tau \in \text{OType} \), let \( \text{TEnv}(\tau) = \{ \varphi \in \text{TVar} \rightarrow_{\text{fin}} T \mid \text{FV}(\varphi) \subseteq \text{dom}(\varphi) \} \). The interpretation of programming-language types is defined by induction, as a function
\[ [\cdot] : \prod_{\tau \in \text{OType}} \text{TEnv}(\tau) \rightarrow T. \]  
We show some cases of the definition here; the complete definition can be found in Appendix A.2.

\[ [\alpha] \varphi = \varphi(\alpha) \]
\[ [\tau_1 \times \tau_2] \varphi = \lambda w. \{ (v_1, v_2) \mid v_1 \in [\tau_1] \varphi(w) \land v_2 \in [\tau_2] \varphi(w) \} \]
\[ [\text{ref } \tau] \varphi = \lambda w. \{ l \mid l \in \text{dom}(w) \land \forall w_1 \geq w. \text{app}(w(l))(w_1) = \triangleright([\tau] \varphi)(w_1) \} \]
\[ [\forall \alpha. \tau] \varphi = \lambda w. \{ \Lambda \alpha.t \mid \forall v \in T, \forall w \geq w. \text{app}(w)(t)(w_1) = \triangleright([\tau] \varphi(\alpha \rightarrow v))(w_1) \} \]

\[ [\mu \alpha. \tau] \varphi = \text{fix } \{ \lambda \alpha.l.w. \{ \text{fold } v \mid \triangleright(v \in [\tau] \varphi(\alpha \rightarrow v)(w)) \} \} \]
\[ [\tau_1 \rightarrow \tau_2] \varphi = \lambda w. \{ \tilde{x}.t \mid \forall w_1 \geq w. \forall v \in [\tau_1] \varphi(w_1), t[w/x] \in \text{comp}([\tau_2] \varphi(\tau_1)(w)) \} \]

Here the operations \( \text{comp} : T \rightarrow T^* \) and \( \text{states} : W \rightarrow P(\text{Store}) \) are given by
\[ \text{comp}(\nu)(v) = \{ t \mid \forall s \in \text{states}(w). \text{eval}(t, s, \lambda(v_1, s_1). \exists w_1 \geq w. v_1 \in \nu(w_1) \land s_1 \in \text{states}(w_1) \} \]
\[ \text{states}(w) = \{ s \mid \text{dom}(s) = \text{dom}(w) \land \forall l \in \text{dom}(w). s(l) \in \text{app}(w(l))(w) \}. \]

Notice that this definition is almost as simple as an attempt at a naive, set-theoretic definition, except for the two explicit uses of \( \triangleright \). In the definition of \( [\mu \alpha. \tau] \), the use of \( \triangleright \) ensures that the fixed point is well-defined according to Theorem 2.7. As for the definition of \( [\text{ref } \tau] \), the \( \triangleright \) is needed because we have \( \triangleright \) instead of the identity in Lemma 3.3. In both cases, the intuition is the usual one from step-indexing: since an evaluation step takes a unit of time, it suffices that a certain formula only holds later.

**Proposition 3.4 (Fundamental theorem).** If \( \vdash t : \tau \), then for all \( w \in W \) we have \( t \in \text{comp}([\tau][\emptyset](w)) \).

**Proof.** To show this, one first generalizes to open types and open terms in the standard way, and then one shows semantic counterparts of all the typing rules of the language. See Appendix A.3. To illustrate the use of \( \triangleright \), we outline the case of reference lookup: \( \vdash !t : \tau \). Here the essential proof obligation is that \( v \in [\text{ref } \tau][\emptyset](w) \) implies \( !v \in \text{comp}([\tau][\emptyset](w)) \). To show this, we unfold the definition of \( \text{comp} \). Let \( s \in \text{states}(w) \) be given; we must show
\[ \text{eval}(!v, s, \lambda(v_1, s_1). \exists w_1 \geq w. v_1 \in [\tau][\emptyset](w_1) \land s_1 \in \text{states}(w_1)). \]  

By the assumption that \( v \in [\text{ref } \tau][\emptyset](w) \), we know that \( v = l \) for some location \( l \) such that \( l \in \text{dom}(w) \) and \( \text{app}(w(l))(w_1) = \triangleright([\tau][\emptyset](w_1)) \) for all \( w_1 \geq w \). Since \( s \in \text{states}(w) \), we know that \( l \in \text{dom}(s) = \text{dom}(w) \) and \( s(l) \in \text{app}(w(l))(w) \). We therefore have \( \text{step}(!v, s, s(l), s) \). Hence, by unfolding the definition of \( \text{eval} \) in (6) and using the rules from Proposition 2.5, it remains to show that \( \exists w_1 \geq w. \triangleright(s(l) \in [\tau][\emptyset](w_1)) \land \triangleright(s \in \text{states}(w_1)) \). We choose \( w_1 = w \). First, \( s \in \text{states}(w) \) and hence \( \triangleright(s \in \text{states}(w)) \). Second, \( s(l) \in \text{app}(w(l))(w) = \triangleright([\tau][\emptyset](w)) \), which means exactly that \( \triangleright(s(l) \in [\tau][\emptyset](w)) \). \( \square \)

### 3.5 The view from the outside

We now return to the standard universe of sets and give external interpretations of the internal results above. One

---

7
basic ingredient is the fact that the constant-presheaf functor \( \Delta : \text{Set} \to \mathcal{S} \) commutes with formation of \( W \)-types. This fact can be shown by inspection of the concrete construction of \( W \)-types for presheaf categories given in [23].

Let \( \text{OType}' \) and \( \text{OTerm}' \) be the sets of possibly open types and terms, respectively, as defined by the grammars above. Similarly, let \( \text{Value}', \text{Store}', \text{Config}' \), and \( \text{step}' \) be the external counterparts of the definitions from the previous sections.

**Proposition 3.5.** \( \text{OType} \cong \Delta(\text{OType}') \), and similarly for \( \text{OTerm, Value, Store, and Config} \). Moreover, under these isomorphisms \( \text{step} \) corresponds to \( \Delta \text{step}' \) as a subobject of \( \text{Config} \times \text{Config} \).

This result essentially says that the external interpretation of the step relation is world-independent, and has the expected meaning: for all \( n \) we have that \( n \models \text{step}((t', s'), (t', s')) \) holds iff \( (t, s) \) actually steps to \( (t', s') \) in the standard operational semantics. We next consider the eval predicate:

**Proposition 3.6.** \( n \models \text{eval}(t, s, Q) \) iff the following property holds: for all \( m < n \), if \( (t, s) \) reduces to \( (v, s') \) in \( m \) steps, then \( (n - m) \models Q(v, s') \).

Using this property and the forcing semantics from Section 2.1, one obtains that the external meaning of the interpretation of types is a step-indexed model in the standard sense. In particular, note that an element of \( \mathcal{P}(\text{Value})(n) \) can be viewed as a set of pairs \( (m, v) \) of natural numbers \( m \leq n \) and values which is downwards closed in the first component.

### 3.6 Discussion

For simplicity, we have just considered a unary model in this extended example; we believe the approach scales well to both relational models and also to more sophisticated models for reasoning about local state [2, 6, 10]. In particular, we have experimented with an internal-logic for-
Lemma 4.2. The functor $\triangleright_1: S/I \to S/I$ is strong and locally contractive.

Lemma 4.3. Let $F: C \to D$, $G: D \to E$ be $S/I$-enriched functors. If either $F$ or $G$ is locally contractive, so is $GF$.

For the statement of the general solutions to recursive domain equations with parameters recall the symmetrization $F: (C^{op} \times C)^n \to C^{op} \times C$ of a functor $F: (C^{op} \times C)^n \to C$ defined as $F(\vec{X},\vec{Y}) = (F(\vec{Y},\vec{X}), F(\vec{X},\vec{Y}))$.

Theorem 4.4. Let $F: ((S/I)^{op} \times S/I)^{n+1} \to S/I$ be locally contractive in the $(n+1)$st variable pair. Then there exists a unique (up to isomorphism) $\text{Fix } F: ((S/I)^{op} \times S/I)^n \to S/I$ such that $F \circ (\text{id}, \text{Fix } F) \cong \text{Fix } F$. Moreover, if $F$ is locally contractive in all variables, so is $\text{Fix } F$.

One can prove that the fixed points obtained by Theorem 4.4 are initial algebras in the sense of Freyd [13–15]. This universal property generalises initial algebras and final coalgebras to mixed-variance functors, and can be used to prove mixed induction / coinduction principles [25].

The formation of recursive types is well-behaved wrt. substitution:

Proposition 4.5. If

$$(S/I)^{op} \times S/I)^{n+1} \xrightarrow{u^*} S/I$$

commutes up to isomorphism, so does

$$(S/I)^{op} \times S/I)^n \xrightarrow{u^*} S/I$$

$$(S/J)^{op} \times S/J)^n \xrightarrow{u^*} S/J$$

Function types. We now define a collection of dependent types $A$ that induce locally contractive strong functors on slices. First, consider the following grammar, in which $C$ and $I$ range over arbitrary types.

$A(\vec{X},\vec{X}) := \vec{X} \mid \text{such that } A(\vec{X},\vec{X}) \times A(\vec{X},\vec{X}) \mid A(\vec{X},\vec{X}) \to A(\vec{X},\vec{X}) \mid \prod_{i: I} A(\vec{X},\vec{X}) \mid \{ a: A(\vec{X},\vec{X}) \mid \phi_{X,X}(a) \} \mid \triangleright A(\vec{X},\vec{X}) \mid \mu X.A((\vec{X}, X), (\vec{X}, X))$$

Note that $X$ is not allowed to appear free in the indexing sets of the dependent sums or products. The type constructor $\mu X.(-)$ corresponds to recursive types.

Any dependent type $\Gamma \vdash A(\vec{X},\vec{X})$ where $\vec{X}, \vec{X}$ do not appear in $\Gamma$, satisfying the grammar above and two further requirements stated below, induces a strong functor

$$[A]: ((\mathcal{E}/[\Gamma])^{op})^n \times (\mathcal{E}/[\Gamma])^m \to \mathcal{E}/[\Gamma]$$

where $n$ is the length of $\vec{X}$ and $m$ is the length of $\vec{X}$.

The strength of this functor is the interpretation of a term of type

$$\Gamma, \vec{f}, \vec{g}, \vec{X}^- \vdash \vec{X}_0^-, \vec{X}_0^+ \vdash \vec{X}_1^- \vdash A(\vec{f}, \vec{g}) : A(\vec{X}_0^-, \vec{X}_0^+) \to A(\vec{X}_1^-, \vec{X}_1^+)$$

and is defined by induction on the structure of $A$. To do this, the type needs to satisfy two requirements. The first is that subset types are only formed for predicates $\phi$ where

$$\phi_{\vec{X}_0^-, \vec{X}_0^+}(a) \to \phi_{\vec{X}_1^-, \vec{X}_1^+}(A(\vec{f}, \vec{g})(a))$$

provably holds for all $\vec{f}, \vec{g}, a$. The second requirement is that the recursive types are only formed in the case where $A((\vec{X}^-, \vec{Y}^-), (\vec{X}^+, \vec{Y}^+))$ is locally contractive in $\vec{Y}^-$ and $\vec{Y}^+$.

Define a functorial type to be a type formed using the grammar above and satisfying the two requirements listed above. We say that a functorial type $A$ is contractive in $X$ if all occurrences of $X$ in $A$ occur under a $\triangleright$. If the functorial type $A$ is contractive, then, by Lemma 4.3, the functor induced by $A$ is locally contractive in the variable corresponding to $X$. In particular, the type formation $\mu X.A$ is valid for all such $A$.

For example, the type $T$ from the previous section is defined by a functorial type.

5. Relation to metric spaces

Let $\text{CBUlt}$ be the category of complete bounded ultrametric spaces and non-expansive maps. In [4–7, 27] we only used those spaces that were also bisected: a metric space is bisected if all non-zero distances are of the form $2^{-n}$ for some natural number $n \geq 0$. Let $\text{BiCBUlt}$ be the full subcategory of $\text{CBUlt}$ of bisected spaces, and let $\text{BiUlt}$ be the category of all bisected ultrametric spaces (necessarily bounded).

Let $tS$ be the full subcategory of $S$ on the total objects.

Proposition 5.1. There is an adjunction between $\text{BiCBUlt}$ and $S$, which restricts to an equivalence between $tS$ and $\text{BiCBUlt}$, as in the diagram:

$$\begin{array}{ccc}
tS & \cong & S \\
\downarrow & & \downarrow \\
\text{BiCBUlt} & \perp & \text{BiUlt}
\end{array}$$
Proof sketch. The functor \( F : \text{BiUlt} \to S \) is defined as follows. A space \((X, d) \in \text{BiUlt}\) gives rise to an indexed family of equivalence relations by \( x =_n x' \Leftrightarrow d(x, x') \leq 2^{-n}\), which can then be viewed as a presheaf: at index \( n \), it is the quotient \( X/(=_n)\), see, e.g. [8]. One can check that \( F \) in fact maps into \( tS \) and that \( F \) has a right adjoint that maps into \( \text{BiCBUlt}\). The left adjoint from \( \text{BiUlt} \) to \( \text{BiCBUlt} \) is then obtained by composition of functors; it is the Cauchy-completion.

**Proposition 5.2.** A morphism in \( \text{BiCBUlt} \) is contractive in the metric sense iff it is contractive in the internal sense of \( S \).

6. Future Work

In this paper we have focused solely on guarded recursion. As future work, it would be interesting to study further the connections between guarded and unguarded recursion in \( S \). For example, it might be possible to show the existence of recursive types in which only negative occurrences of the recursion variable were guarded.

We plan to make a tool for formalized reasoning in the internal logic of \( S \). We have conducted some initial experiments by adding axioms to Coq and used it to formalize some of the proofs from [6] involving recursively defined relations on recursively defined types. These experiments suggest that it will be important to have special support for the manipulation of the isomorphisms involved in recursive type equations, such as the coercions and canonical structures of [16].

Acknowledgments. We thank Andy Pitts and Paul Blain Levy for encouraging discussions.

References


A More details on the application to step-indexing

Here are some more details on the application in Section 3. Everything is this appendix should be understood within the logic of $\mathcal{S}$.

A.1 Language

The full language considered in the application is shown in Figure 1.

A.2 Interpretation of types

Recall that we have

\[
\begin{align*}
\mathcal{W} &= N \to_{\text{fin}} \mathcal{\tilde{T}} \\
\mathcal{T} &= \mathcal{W} \to_{\text{mon}} \mathcal{P}(\text{Value}) \\
\mathcal{T}^c &= \mathcal{W} \to \mathcal{P}(\text{Term})
\end{align*}
\]

and

\[
\text{app} : \mathcal{T} \rightarrow \mathcal{T}, \quad \text{lam} : \mathcal{T} \rightarrow \mathcal{T}
\]

with \(\text{app} \circ \text{lam} = \Rightarrow : \mathcal{T} \rightarrow \mathcal{T}\).

Let \(\text{TVar}\) be the set of type variables, and for \(\tau \in \text{OType}\), let \(\text{TEnv}(\tau) = \{ \varphi \in \text{TVar} \rightarrow_{\text{fin}} \mathcal{T} \mid \text{FV}(\varphi) \subseteq \text{dom}(\varphi) \}\). The interpretation of programming-language types is defined by induction, as a function

\[
[\_] : \prod_{\tau \in \text{OType}} \text{TEnv}(\tau) \rightarrow \mathcal{T}.
\]

Here the operations \(\text{comp} : \mathcal{T} \rightarrow \mathcal{T}^c\) and \(\text{states} : \mathcal{W} \rightarrow \mathcal{P}(\text{Store})\) are given by

\[
\begin{align*}
\text{comp}(\nu)(w) &= \{ t \mid \forall s \in \text{states}(w). \text{eval}(t, s, \lambda(v_1, s_1). \exists w_1 \geq w. \ \nu_1 \in \nu(w_1) \land s_1 \in \text{states}(w_1)) \}
\end{align*}
\]

\[
\begin{align*}
\text{states}(w) &= \{ s \mid \text{dom}(s) = \text{dom}(w) \land \forall l \in \text{dom}(w). s(l) \in \text{app}(w(l))(w) \}. \\
\end{align*}
\]

A.3 Soundness and the fundamental theorem

Given \(\Xi\) and \(\Gamma\) such that \(\Gamma\) is well-formed in \(\Xi\), and given \(\varphi \in \mathcal{T}^\Xi\), define

\[
[\Gamma] \varphi(w) = \{ \rho : \text{Value}^{\text{dom}(\Gamma)} \mid \forall (x, \tau) \in \Gamma. \rho(x) \in [\tau] \varphi(w) \}.
\]

Abbreviate \(\tau^c \varphi = \text{comp}([\tau] \varphi)\).

Now we define semantic validity. The notation

\[
\Xi \mid \Gamma = t : \tau
\]

means: For all \(w \in W\), all \(\varphi \in \mathcal{T}^\Xi\), and all \(\rho \in [\Gamma] \varphi(w)\), we have \(\rho(t) \in [\tau] \varphi(w)\). (Here \(\rho(t)\) is \(\rho\) acting by substitution on \(t\).)

To show the fundamental theorem, we must show semantic counterparts of all the typing rules. First we need some “monadic” properties of the \(\text{comp}\) operator. For \(\nu \in \mathcal{T}\) and \(\xi \in \mathcal{T}^c\) and \(w \in \mathcal{W}\), let \(\nu \rightarrow_w \xi\) be the set of closed evaluation contexts \(E\) that satisfy the following property: for all \(w_1 \geq w\) and \(v \in \nu(w_1)\) we have \(E[v] \in \xi(w_1)\).

Lemma A.1.

1. If \(v \in \nu(w)\), then \(v \in \text{comp}(\nu)(w)\).

2. If \(t \in \text{comp}(\nu_1)(w) \land E \in \nu_1 \rightarrow_w \text{comp}(\nu_2)\), then \(E[t] \in \text{comp}(\nu_2)(w)\).

Proof. The first part follows immediately from the definitions of \(\text{comp}\) and \(\text{eval}\). As for the second part, let \(t \in \text{comp}(\nu_1)(w) \land E \in \nu_1 \rightarrow_w \text{comp}(\nu_2)\) be given; we must show that \(E[t] \in \text{comp}(\nu_2)(w)\). We unfold the definition of \(\text{comp}\). Let \(s \in \text{states}(w)\) be given; we must show that \(\text{eval}(E[t], s, Q)\) where

\[
Q(v_2, s_2) = \exists v_2 \geq w. v_2 \in v_2(w_2) \land s_2 \in \text{states}(w_2).
\]

By Proposition 3.2, it suffices to show

\[
\text{eval}(t, s, \lambda(v_1, s_1). \text{eval}(E[v_1], s_1, Q)). \quad (3)
\]

Since \(t \in \text{comp}(\nu_1)(w) \land s \in \text{states}(w)\) we know that

\[
\text{eval}(t, s, \lambda(v_1, s_1). \exists w_1 \geq w. v_1 \in v_1(w_1) \land s_1 \in \text{states}(w_1)).
\]
Types: $\tau ::= 1 \mid \tau_1 \times \tau_2 \mid 0 \mid \tau_1 + \tau_2 \mid \mu\alpha.\tau \mid \forall\alpha.\tau \mid \alpha \mid \tau_1 \rightarrow \tau_2 \mid \text{ref } \tau$

Terms: $t ::= x \mid l \mid () \mid \langle t_1, t_2 \rangle \mid \text{fst } t \mid \text{snd } t \mid \text{void } t \mid \text{inl } t \mid \text{inr } t \mid \text{case } t_0 x_1 t_1 x_2 t_2 \mid \text{fold } t \mid \text{unfold } t \mid \Lambda\alpha.t \mid t \left[ \tau \right] \mid \lambda x : t. \mid t_1 t_2 \mid \text{fix } f.\lambda x . t \mid \text{ref } t \mid !t \mid t_1 := t_2$

Typing rules:

$$
\frac{\Gamma \vdash x : \tau}{\exists \Gamma \vdash \Gamma \parallel \left( \exists \Gamma \vdash \Gamma \parallel \left( \tau = \tau \right) \right)}
$$

$$
\frac{\Gamma \vdash t_1 : \tau_1 \quad \Gamma \vdash t_2 : \tau_2}{\exists \Gamma \parallel \left( \tau_1 \times \tau_2 \right)}
$$

$$
\frac{\exists \Gamma \vdash \text{fst } t : \tau_1}{\exists \Gamma \parallel \left( \tau_1 \times \tau_2 \right)}
$$

$$
\frac{\exists \Gamma \vdash \text{inl } t : \tau_1 + \tau_2}{\exists \Gamma \vdash t : \tau (i = 1, 2)}
$$

$$
\frac{\exists \Gamma \vdash \text{case } t_0 x_1 t_1 x_2 t_2 : \tau}{\exists \Gamma \vdash t_0 : \tau_1 + \tau_2}
$$

$$
\frac{\exists \Gamma \vdash \text{fold } t : \mu\alpha.\tau}{\exists \Gamma \parallel \mu\alpha.\tau}
$$

$$
\frac{\exists \alpha, \alpha \vdash \Gamma \vdash t : \tau[\mu\alpha.\tau/\alpha]}{\exists \Gamma \parallel \mu\alpha.\tau}
$$

$$
\frac{\exists \Gamma \parallel \Lambda\alpha.t \parallel \forall\alpha.\tau}{\exists \Gamma \parallel \Lambda\alpha.t \parallel \forall\alpha.\tau}
$$

$$
\frac{\exists \Gamma \parallel \text{case } t_0 x_1 t_1 x_2 t_2 : \tau}{\exists \Gamma \parallel \text{ref } t : \text{ref } \tau}
$$

$$
\frac{\exists \Gamma \vdash \text{fix } f.\lambda x . t : \tau_0 \rightarrow \tau_1}{\exists \Gamma \parallel \text{fix } f.\lambda x . t : \tau_0 \rightarrow \tau_1}
$$

$$
\frac{\exists \Gamma \vdash t : \text{ref } \tau}{\exists \Gamma \parallel \text{ref } t : \text{ref } \tau}
$$

$$
\frac{\exists \Gamma \vdash t_1 : \text{ref } \tau \quad \exists \Gamma \vdash t_2 : \tau}{\exists \Gamma \vdash t_1 := t_2 : 1}
$$

Figure 1. Programming language
We can therefore use Proposition 3.1 to show (3): it suffices to show that $\exists w \geq v. v_1 \in \nu_1(w_1) \land s_1 \in states(w_1)$ implies $\text{eval}(E[v_1], s_1, Q)$. So let $w_1 \geq w$ be given and assume that $v_1 \in \nu_1(w_1)$ and $s_1 \in states(w_1)$. Then, since $E \in v_1 \in comp(v_2)$, we have $E[v_1] \in comp(v_2)(w_1)$ and hence

$$\text{eval}(E[v_1], s_1, \lambda(v_2, s_2), \exists w_2 \geq w_1.
\quad v_2 \in \nu_2(w_2) \land s_2 \in states(w_2)).$$

Since $w_1 \geq w$, another use of Proposition 3.1 gives $\text{eval}(E[v_1], s_1, Q)$, which is what we had to show. □

Proof of Proposition 3.4 (fundamental theorem). We show four key cases.

Case “allocation”:
If $\exists | \Gamma \models t : \tau$, then $\exists | \Gamma \models \text{ref } t : \text{ref } \tau$.

Let $w \in W$ and $\varphi \in \mathcal{T}^= $ and $\rho \in \Gamma[\varphi]$ be given; we must show that $\rho(\text{ref } t) \in \Gamma[\text{ref } \tau[\varphi](w)]$. Since $\exists | \Gamma \models t : \tau$ holds we know that $\rho(t) \in \Gamma[\tau[\varphi](w)]$. Therefore, by Lemma A.1, it suffices to show that $\text{ref } t \in \Gamma[\tau[\varphi](w)]$. To that end, let $w_1 \geq w$ and $v \in \Gamma[\varphi(w_1)]$ be given. We must show that $v \in \Gamma[\text{ref } \tau[\varphi](w_1)]$.

Let $s \in states(w_1)$ be given. By definition of $\text{comp } w_1$ we must show

$$\text{eval}(v, s, \lambda(v_2, s_2), \exists w_2 \geq w_1.
\quad v_2 \in \Gamma[\text{ref } \tau][\varphi(w_2)] \land s_2 \in states(w_2)).$$

Let $l$ be the smallest location not in $s$. Then we have $\text{step}(\text{ref } v, s, (l, s_1))$ where $s_1 = s[l \mapsto v]$. Therefore, by definition of $\text{eval}$ and Proposition 2.5, it suffices to show

$$\exists w_2 \geq w_1. l \in \Gamma[\text{ref } \tau][\varphi(w_2)] \land s_2 \in states(w_2)).$$

(In fact, we are only required to show $\supset$ applied to this formula, which is weaker by Proposition 2.5(1).) To that end, we choose $w_2 = w_1[l \mapsto \text{lam}(\Gamma[\tau][\varphi])]$. It remains to show

$$l \in \Gamma[\text{ref } \tau][\varphi(w_2)] \land s_2 \in states(w_2)).$$

As for (4), we expand the definition of $\Gamma[\text{ref } \tau]$. Clearly we have $l \in \text{dom}(w_2)$ as required. Now let $w_3 \geq w_2$ be given; Lemma 3.5 gives

$$\text{app}(w_2(l))(w_3) = \text{app}(\text{lam}(\Gamma[\tau][\varphi]))(w_3)$$

as required.

As for (5), we first have that $\text{dom}(s_1) = \text{dom}(w_2)$ since $s \in states(w_1)$. Second, we must show that $s_1(l) \in \text{app}(w_2(l'))(w_2)$ for all $l' \in \text{dom}(s_1)$. For $l' = l$ we have

$$\text{app}(w_2(l))(w_2) = \supset(\Gamma[\tau][\varphi](w_2))$$

as above. But $s_1(l) = v$, and we know that $v \in \Gamma[\tau][\varphi](w_1)$ where

$$\Gamma[\tau][\varphi](w_1) \subseteq \Gamma[\tau][\varphi](w_2) \subseteq \supset(\Gamma[\tau][\varphi](w_2))$$

by monotonicity and Proposition 2.5(1). We conclude that $s_1(l) \in \text{app}(w_2(l))(w_2)$.

For $l' \neq l$ we have $s_1(l') = s(l')$. Since $s \in states(w_1)$ we know that $s(l') \in \text{app}(w_1(l'))(w_1)$. But

$$\text{app}(w_1(l'))(w_1) = \text{app}(w_2(l'))(w_2) \subseteq \text{app}(w_2(l'))(w_2)$$

by monotonicity. Therefore $s_1(l') \in \text{app}(w_2(l'))(w_2)$, which completes the proof of (5).

Case “lookup”:
If $\exists | \Gamma \models t : \text{ref } \tau$ then $\exists | \Gamma \models ! t : \tau$.

Let $w \in W$ and $\varphi \in \mathcal{T}^= $ and $\rho \in \Gamma[\varphi]$ be given; we must show that $\rho(!t) \in \Gamma[\text{ref } \tau \varphi](w)$. Since $\exists | \Gamma \models t : \text{ref } \tau$ holds we know that $\rho(t) \in \Gamma[\text{ref } \tau \varphi](w)$. Therefore, by Lemma A.1, it suffices to show that $!t \in \Gamma[\text{ref } \tau \varphi](w)$. This is essentially what was done in the proof sketch in the main text, but for completeness we repeat the argument here.

Let $w_1 \geq w$ and $v \in \Gamma[\text{ref } \tau \varphi](w_1)$ be given. We must show that $!v \in \text{comp}(\Gamma[\tau \varphi](w_1))$. We unfold the definition of $\text{comp } w_1$ and must show

$$\text{eval}(!v, s, \lambda(v_2, s_2), \exists w_2 \geq w_1.
\quad v_2 \in \Gamma[\tau \varphi](w_2) \land s_2 \in states(w_2)).$$

By the assumption that $v \in \Gamma[\text{ref } \tau \varphi](w_1)$, we know that $v = l$ for some location $l$ such that $l \in \text{dom}(w_1)$ and $\text{app}(w_1(l))(w_2) = \supset(\Gamma[\tau \varphi](w_2))$ for all $w_2 \geq w_1$. Since $s \in states(w_1)$, we know that $l \in \text{dom}(s) = \text{dom}(w_1)$ and $s(l) \in \text{app}(w_1(l))(w_1)$. We therefore have $\text{step}((!v, s, (l', s)), (l, s))$. Hence, by unfolding the definition of $\text{eval}$ in (6) and using the rules from Proposition 2.5, it remains to show that

$$\exists w_2 \geq w_1. \supset(s(l) \in \Gamma[\tau \varphi](w_2)) \land \supset(s \in states(w_2)).$$

To that end, choose $w_2 = w_1$. First, $s \in states(w_1)$ and hence $\supset(s \in states(w_1))$. Second,

$$s(l) \in \text{app}(w_1(l))(w_1) = \supset(\Gamma[\tau \varphi](w_1)),$$

which means exactly that $\supset(s \in \Gamma[\tau \varphi](w_1))$.

Case “assignment”:
If $\exists | \Gamma \models t_1 : \text{ref } \tau$ and $\exists | \Gamma \models t_2 : \tau$, then $\exists | \Gamma \models t_1 := t_2 : 1$.

Here we must use Lemma A.1 twice. Let $w \in W$ and $\varphi \in \mathcal{T}^= $ and $\rho \in \Gamma[\varphi] be given; we must show that

$$\rho(t_1 := t_2) \in \Gamma[\varphi].$$
Since $\Xi \models \Gamma \models t_1 : \text{ref} \tau$ holds we know that $\rho(t_1) \in [\text{ref} \tau]^\omega \varphi(w)$. Therefore, by Lemma A.1, it suffices to show that
\[
(- := \rho(t_2)) \in [\text{ref} \tau] \varphi \sto w [1]^\omega \varphi.
\]
So let $w_1 \geq w$ and $v_1 \in [\text{ref} \tau] \varphi(w_1)$ be given; we must show that $(v_1 := \rho(t_2)) \in [1]^\omega \varphi(w_1)$. By assumption we have $\rho(t_2) \in [\text{ref} \tau] \varphi(w_1)$, so by Lemma A.1 again, it suffices to show that
\[
(v_1 := -) \in [\text{ref} \tau] \varphi \sto w [1]^\omega \varphi.
\]
Therefore, let $w_2 \geq w_1$ and $v_2 \in [\tau] \varphi(w_2)$ be given. The final proof obligation is to show that
\[
(v_1 := v_2) \in [1]^\omega \varphi(w_2).
\]
We unfold the definition of $\text{comp}$. Assume that $s \in \text{states}(w_2)$ is given; we must show
\[
eval((v_1 := v_2), s, \lambda(v_3, s_3). \exists w_3 \geq w_2.
\]
\[
v_3 \in [1] \varphi(w_3) \land s_3 \in \text{states}(w_3).
\]
By monotonicity we have $v_1 \in [\text{ref} \tau] \varphi(w_2)$, and therefore $v_1 = l$ for some $l \in \text{dom}(w_2)$ such that
\[
\text{app}(w_2(l))(w_3) \overset{[\tau] \varphi(w_3)}{\Rightarrow} \text{for all } w_3 \geq w_2. \quad (7)
\]
Furthermore, since $s \in \text{states}(w_2)$ we know that $\text{dom}(s) = \text{dom}(w_2)$ and hence $l \in \text{dom}(s)$. Therefore $\text{step}((v_1 := v_2, s), ((l, s[l \mapsto v_2])))$ holds. By definition of $\text{eval}$ and Proposition 2.5, it then suffices to show
\[
\exists w_3 \geq w_2. () \in [1] \varphi(w_3) \land s[l \mapsto v_2] \in \text{states}(w_3).
\]
We choose $w_3 = w_2$. Now $() \in [1] \varphi(w_2)$ holds trivially, and it remains to show that $s[l \mapsto v_2] \in \text{states}(w_2)$. For $l' \neq l$ we have
\[
(s[l \mapsto v_2])(l') = s(l') \in \text{app}(w_2(l'))(w_2)
\]
since $s \in \text{states}(w_2)$. Furthermore,
\[
(s[l \mapsto v_2])(l) = v_2 \in [\tau] \varphi(w_2),
\]
and therefore $\overset{[\tau] \varphi(w_2)}{\Rightarrow} \text{by Proposition 2.5(1)}. \text{But this means exactly that } v_2 \in \overset{[\tau] \varphi(w_2)}{\Rightarrow}$. We conclude from (7) that $v_2 \in \text{app}(w_2(l))(w_2)$ as required.

Case “unfold”:
If $\Xi \models \Gamma \models t : \mu \alpha. \tau$, then $\Xi \models \Gamma \models \text{unfold } t : \tau(\mu \alpha. \tau)/\alpha$.

Abbreviate $\tau_1 = \tau(\mu \alpha. \tau)/\alpha$. Let $w \in \mathcal{W}$ and $\varphi \in \mathcal{T}^\Xi$ and $\rho \in [\Gamma] \varphi$ be given; we must show that $\rho(\text{unfold } t) \in [\tau_1]^\omega \varphi(w)$. Since $\Xi \models \Gamma \models t : \mu \alpha. \tau$ holds we know that $\rho(t) \in [\mu \alpha. \tau]^\omega \varphi(w)$. Therefore, by Lemma A.1, it suffices to show that unfold $\rho \in [\mu \alpha. \tau]^\omega \varphi \st w [\tau_1]^\omega \varphi$. To that end, let $w_1 \geq w$ and $v \in [\mu \alpha. \tau]^\omega \varphi(w_1)$ be given. We must show that $\text{unfold } v \in [\tau_1]^\omega \varphi(w_1)$.

Let $s \in \text{states}(w_1)$ be given. By definition of $\text{comp}$ we must show
\[
\text{eval}(\text{unfold } v, s, \lambda(v_1, s_1) \exists w_2 \geq w_1.
\]
\[
v_1 \in [\tau_1] \varphi(w_2) \land s_1 \in \text{states}(w_2). \quad (8)
\]
By definition of $[\mu \alpha. \tau] \varphi$ we know that $v = \text{fold } v_0$ for some $v_0$ such that $\overset{[\tau] \varphi(a \mapsto \mu \alpha. \tau] \varphi(w_1))}{\Rightarrow}$. By Proposition 2.5 and a substitution lemma (shown by an easy induction on types), this means that $\overset{[\tau] \varphi(w_1)}{\Rightarrow}$. Since $v = \text{fold } v_0$ we have $\text{step}(\text{unfold } v, s), (v_0, s))$. Therefore, by unfolding the definition of $\text{eval}$ in (8) and using Proposition 2.5, it suffices to show
\[
\exists w_2 \geq w_1. \overset{[\tau] \varphi(w_2)}{\Rightarrow} \text{and } \overset{(s \in \text{states}(w_2))}{\Rightarrow}.
\]
We choose $w_2 = w_1$. We have already shown that $\overset{[\tau] \varphi(w_1)}{\Rightarrow}$ holds, and Proposition 2.5(1) gives that $s \in \text{states}(w_1)$ implies $\overset{(s \in \text{states}(w_1))}{\Rightarrow}$, as required.

As an immediate corollary of the fundamental theorem we get a type-safety result for the “temporal” semantics given by the eval predicate. This is formulated by means of a trivial post-condition.

**Corollary A.2 (Type safety).** Assume that $\vdash t : \tau$ holds. Then $\text{eval}(t, s_{\text{init}}, \top)$ holds where $s_{\text{init}}$ is the empty store.

**Proof.** Follows directly from the fundamental theorem (using the empty world $\emptyset \in \mathcal{W}$) and Proposition 3.1.

**B Solving recursive domain equations**

This section contains proofs of Theorem 4.4 and Proposition 4.5. We first consider covariant domain equations.

**Definition B.1.** Let $\mathcal{C}, \mathcal{D}$ be $\mathcal{S}/\mathcal{I}$-enriched categories.

1. We say that a $\mathcal{S}/\mathcal{I}$-enriched functor $F: \mathcal{C} \to \mathcal{D}$ is locally contractive if each $F_{X,Y}: \text{Hom}_\mathcal{C}(X,Y) \to \text{Hom}_\mathcal{D}(FX,FY)$ factors through $\text{next}_{\text{Hom}_\mathcal{C}(X,Y)}$.

2. We say that $\mathcal{C}$ is contractively complete if for any locally contractive functor $F: \mathcal{C} \to \mathcal{C}$ there exists an $X$ such that $F(X) \cong X$.

Recall that the slice categories are $\mathcal{S}$-enriched: given objects $p_Y: Y \to I, p_Z: Z \to I$ in $\mathcal{S}/\mathcal{I}$ we form the hom-object as the equalizer
\[
\text{Hom}_{\mathcal{S}/\mathcal{I}}(p_Y, p_Z) \longrightarrow [Y \to Z] \xrightarrow{p_Z \circ (-)} [Y \to I]
\]
Alternatively, $\text{Hom}_{\mathcal{S}/\mathcal{I}}(p_Y, p_Z)$ can be defined in the internal language of $\mathcal{S}$ as $\prod_{i: I} Z_i^i$. 

14
Theorem B.2. Let $\mathbb{C}$ be a complete and $S$-enriched category. Then $\mathbb{C}$ is contractively complete. In particular, every slice $S/I$ is contractively complete in the $S$-enriched sense.

In the theorem we mean $\mathbb{C}$ is complete in the usual, not the enriched sense.

Proof (outline). The proof is based on previous work on recursive domain equations in CBUlt-enriched categories. In Birkedal et al.5 (BST) it is proved that any complete CBUlt-enriched category with non-empty hom-sets has solutions to all recursive domain equations given by locally contractive (in the metric space sense) functors. We can relate this result to the current setting, because the product preserving functor $G: S \rightarrow BiCBUlt$ mentioned in Section 5 allows us to consider any $S$-enriched category as a CBUlt-enriched category with hom-objects $G(\text{Hom}_{\mathbb{C}}(A, B))$. (In more detail, the functor $G$ maps a presheaf $X : \omega^{op} \rightarrow \text{Set}$ to its limit $\lim X$ in $\text{Set}$, equipped with the following metric: $d(x, x') = \inf \{2^{-n} \mid \pi_n(x) = \pi_n(x')\}$. Here the $\pi_n : \lim X \rightarrow X(n)$ are the maps from the limiting cone.) If $F$ is locally contractive in the $S$-enriched sense then it is also locally contractive in the metric space sense.

However, we cannot simply lift the results of BST because there is no guarantee that $\mathbb{C}$ considered as a CBUlt-enriched category has non-empty hom-objects. In BST the solutions to recursive domain equations are constructed using limit/colimit coincidence of chains of embedding/projection pairs, and the assumption of non-empty hom-sets is used to get the sequence started via an embedding/projection pair from 1 to $F(1, 1)$.

For the proof of Theorem B.2, since we only consider covariant functors, we can consider the sequence

$$F^1 \xleftarrow{p} F^2 \xrightarrow{\text{proj}} F^3 \xrightarrow{\text{proj}} F^4 \xrightarrow{\text{proj}} \ldots$$

as constructed in Section 2.2 instead. It is an easy diagram chase to check that $pe$ is the identity and so this really is a sequence of injection/projection pairs. Taking $X$ to be the limit of the maps $F^i p$ the proof of Section 3 of BST can be reused to show that $F(X) \cong X$.

Lemma B.3. If $F : S/I \rightarrow S/I$ is $S/I$-enriched then it is also $S$-enriched. If moreover $F$ is locally contractive in the $S/I$-enriched sense, then it is also locally contractive in the $S$-enriched sense.

Proof. For $F$ to be $S/I$-enriched means that there exists a morphism $Y^X \rightarrow F(Y)^{F(X)}$ in $S/I$ inducing the action of $F$ on morphisms. From this one can construct an $S$-enrichment as the composition

$$\text{Hom}_{S/I}(X, Y) \cong \text{Hom}_{S/I}(1, Y^X) \rightarrow \text{Hom}_{S/I}(1, F(Y)^{F(X)}) \cong \text{Hom}_{S/I}(F(X), F(Y)).$$

If $F$ is locally contractive in the $S/I$-enriched sense, then we can factor the action on morphisms through next as follows

$$\text{Hom}_{S/I}(X, Y) \xrightarrow{\text{next}} \text{Hom}_{S/I}(X, Y) \xrightarrow{\text{next}} \text{Hom}_{S/I}(1, Y^X) \rightarrow \text{Hom}_{S/I}(1, F(Y)^{F(X)}) \cong \text{Hom}_{S/I}(F(X), F(Y)).$$

using naturality of next, local contractiveness of $\text{next}$, and that $\text{next} \circ 1 \cong 1$.

Theorem B.4. $S/I$ is contractively complete as an $S/I$-enriched category.

Proof. Suppose $F : S/I \rightarrow S/I$ is locally contractive in the $S/I$-enriched sense. By Lemma B.3 it is also locally contractive in the $S$-enriched sense, and so by Theorem B.2 it has a fixed point.

Just like fixed points on the term level are unique, so are fixed points on the type level unique.

Lemma B.5. Let $\mathbb{C}$ be $S/I$-enriched and let $F : \mathbb{C} \rightarrow \mathbb{C}$ be a locally contractive functor. If $X \cong F(X)$, then the two directions of the isomorphism give an initial algebra structure and a final coalgebra structure for $F$ on $X$. In particular, if $F(X) \cong X$ and $F(Y) \cong Y$, then $X \cong Y$.

Proof. Given an isomorphism $f : FX \rightarrow X$ and some other algebra $g : FZ \rightarrow Z$, there is an algebra homomorphism iff the diagram

$$\begin{array}{ccc}
FZ & \xrightarrow{Fh} & FX \\
\downarrow{f^{-1}} & & \downarrow{f} \\
X & \rightarrow & Z
\end{array}$$

commutes, i.e., iff $h$ is a fixed point of the map $h \mapsto g \circ F(h) \circ f^{-1}$, which is a contractive endomorphism on $\text{Hom}_{\mathbb{C}}(X, Z)$. Since this map has exactly one fixed point we conclude that there is exactly one algebra homomorphism from $f$ to $g$.

There is also a morphism in the topos computing the unique mediating homomorphism from the initial algebra.
Lemma B.6. Let $C$ and $F$ be as in Lemma B.5, and let $f: FX \to X$ be an isomorphism. For any $Z$ there exists a morphism $k: \text{Hom}_C(FZ, Z) \to \text{Hom}_C(X, Z)$ such that $\forall g: \text{Hom}_C(FZ, Z), k(g) \circ f = g \circ F(k(g))$ holds in the internal language of $S/I$. 

Proof. Define $k$ to be the fixed point of the map $\text{Hom}_C(FZ, Z) \times \text{Hom}_C(X, Z) \to \text{Hom}_C(X, Z)$ mapping $g, h$ to $g \circ Fh \circ f^{-1}$.

Lemma B.7. Let $C, \mathbb{D}$ be $S/I$-enriched categories and let $F: \mathbb{D} \times C \to C$ be enriched and locally contractive in the second variable. If the functor $F(X, -): C \to C$ has an initial algebra for all $X$ in $\mathbb{D}$, then there is a $S/I$-enriched functor $\mu F: \mathbb{D} \to C$ mapping $X$ to the initial algebra. If, moreover, $F$ is locally contractive in the first variable, then $\mu F$ is locally contractive.

Proof. The functor $\mu F$ is defined (as is standard) to map $f: d \to d'$ to the unique $\mu F(f)$ making the diagram

\[
\begin{array}{ccc}
F(d, \mu F(d)) & \longrightarrow & \mu F(d) \\
F(d, \mu F(f)) \downarrow & & \mu F(f) \\
F(d, \mu F(d')) & \xrightarrow{F(f, \text{id})} & F(d', \mu F(d')) \longrightarrow \mu F(d')
\end{array}
\]

commute. Now, the strength of $\mu F$ is obtained by composing the morphism $\text{Hom}_{\mathbb{D}}(d, d') \to \text{Hom}_C(F(d, \mu F(d')), \mu F(d'))$ mapping $f$ to the composite in the bottom line of (9) with the morphism of Lemma B.6. In the case of $F$ being locally contractive in both variables, the first stage of this composite morphism is contractive and so $\mu F$ becomes locally contractive.

Now we turn to general recursive domain equations. Suppose $G: (C^{op} \times C)^n \to C$. Define the symmetrization of $G$ denoted $\tilde{G}: (C^{op} \times C)^n \to C^{op} \times C$ as $\tilde{G}(\tilde{X}, \tilde{Y}) = (G(\tilde{Y}, \tilde{X}), G(\tilde{X}, \tilde{Y}))$. Freyd [13–15] argues that the natural generalisation of initial algebras for covariant functors to mixed variance functors such as $G$ (in the case of $n = 1$) are initial algebras for $\tilde{G}$ which he calls initial dialgebras for $G$.

Lemma B.8. Let $C$ be $S/I$-enriched and $G: C^{op} \times C \to C$ be a locally contractive functor. If $G(X, Y) \cong Y$ and $G(Y, X) \cong X$ then the pair $(X, Y)$ together with the isomorphisms constitute an initial dialgebra for $G$. In particular, $(X, Y)$ is unique up to isomorphism with this property. Moreover $X \cong Y$.

Proof. If $G$ is locally contractive, so is $\tilde{G}$, and Lemma B.5 applies proving that $(X, Y)$ is an initial dialgebra. To show $X \cong Y$ note that the hypothesis of the lemma is symmetric in $X$ and $Y$, so we may apply what we have just proved to conclude that $(Y, X)$ is an initial dialgebra. By uniqueness of initial dialgebras $(X, Y) \cong (Y, X)$.

We can now give the promised proofs of theorems from the main text.

Proof of Theorem 4.4. Consider first the case of $n = 0$. Recall the functor $\mu F: C^{op} \to C$ from Lemma B.7 mapping $X$ to the unique fixed point of $F(X, -)$. Define $Z$ to be the unique fixed point of the functor $X \to \mu F(X, X)$ and define $W = \mu F(Z)$. Then $F(W, Z) = \mu F(Z, Z) \cong Z$ and $F(Z, W) = \mu F(Z, Z) \cong W$, and so Lemma B.8 applies giving the unique solution to $F$ and proving that $W \cong Z$.

In the general case of $n \neq 0$, Lemma B.7 applies to give the functor $\text{Fix} F$.

Proof of Proposition 4.5. Recall that $\text{Fix} G \circ u^*$ is unique up to isomorphism such that

$$G(u^*(\tilde{X}, \tilde{Y}), \tilde{G}(u^*(\tilde{X}, \tilde{Y}))) \cong \tilde{G}(u^*(\tilde{X}, \tilde{Y})).$$

Now,

$$G(u^*(\tilde{X}, \tilde{Y}), u^*(\tilde{G}(\tilde{X}, \tilde{Y}))) \cong u^*(F(\tilde{X}, \tilde{Y}, \tilde{G}(\tilde{X}, \tilde{Y}))) \cong u^*\text{Fix} F(\tilde{X}, \tilde{Y})$$

and so we conclude $u^*\text{Fix} F(\tilde{X}, \tilde{Y}) \cong \text{Fix} G(u^*(\tilde{X}, \tilde{Y}))$. 

\[\square\]