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 $\mathcal{S}$ of trees as a model of guarded recursion. We study the internal dependently-typed higher-order logic of $\mathcal{S}$ and show that $\mathcal{S}$ 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 $\mathcal{S}$ 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 $\mathcal{S}$.**

I. 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 $\mathcal{S}$, 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 $\mathcal{S}$ 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 $\mathcal{S}$ 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 $\mathcal{S}$-logic, we present a model of $\mathsf{F}_{\mu,\mathsf{ref}}$, 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 $\mathcal{S}$ is the category of presheaves on ω, 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 $\mathcal{S}$ is a contravariant functor from ω (viewed as a preorder) to $\mathbf{Set}$. We think of X as a variable set, i.e., a family of sets $X(n)$, indexed over natural numbers n, and with restriction maps $X(n{+}1) \to X(n)$. Morphisms $f : X \to Y$ are natural transformations from X to Y. The variable sets include the ordinary sets as so-called constant sets: for an ordinary set S, there is an object $\Delta(S)$ in $\mathcal{S}$ with $\Delta(S)(n) = S$ for all n. Since $\mathcal{S}$ is a category of presheaves, it is a topos, i.e., a cartesian closed category with a subobject classifier Ω (a type of propositions). The internal logic of $\mathcal{S}$ 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 ω) 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 φ 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 φ 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 Löb rule for induction, $(\triangleright\varphi \to \varphi) \to \varphi$, 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 Löb rule, and well-defined guarded recursive definitions of predicates.

Moreover, there is also a later operator $\blacktriangleright$ (a functor) on the variable sets themselves: $\blacktriangleright(X)$ is given by $\blacktriangleright(X)(1) = \{\star\}$ and $\blacktriangleright(X)(n+1) = X(n)$. 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 $\blacktriangleright$. Intuitively, such a recursively specified variable set is well-defined since by definition of $\blacktriangleright$, 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 $\mathsf{F}_{\mu,\mathsf{ref}}$. 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 $\overline{\omega}$, where $\overline{\omega}$ is the complete Heyting algebra of natural numbers with the usual ordering and extended with a top element ∞. 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 $\mathbf{Set}(\_,\overline{\omega})$. 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 [23] which can be modelled using complete bounded ultrametric spaces [5]. We show in Section V that the category $\mathit{BiCBUlt}$ 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 [25] 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, $\mathit{BiCBUlt}$ 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 III. 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 $\mathbf{Set}(\_,\overline{\omega})$ 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 (variable sets rather than only constant sets), which makes it possible also to have mixed-variance recursively defined types.[3]

Dreyer *et al.* developed an extension of LSLR called LADR for reasoning about step-indexed models of the programming language $\mathsf{F}_{\mu,\mathsf{ref}}$ 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 III, we *define* a set of worlds inside the $\mathcal{S}$ 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].

[1] Recall that the tripos $\mathbf{Set}(\_,\overline{\omega})$ is a model of logic in which types and terms are interpreted as sets and functions, and predicates are interpreted as $\overline{\omega}$-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.

[3] The terminology can be slightly confusing: in [3], our notion of recursive relations were called recursive types, probably because the authors of *loc.cit.* used such to interpret recursive types of a programming language. Recursive types in our sense were not considered in [3].

The difference is that we allow for general mixed-variance recursive types, but on the other hand we require that all occurrences of recursion variables must be guarded.

Summary of contributions: We show how the topos $\mathcal{S}$ provides a simple but powerful model of guarded recursion, allowing for guarded recursive definitions of both terms and types in the internal dependently-typed higher-order logic. In particular, we

- show that the two later modalities are well-behaved on slices;
- give existence theorems for fixed points of guarded recursive terms and guarded nested dependent mixed-variance recursive types;
- present, as an example application, a synthetic model of $\mathsf{F}_{\mu,\mathsf{ref}}$ constructed internally in $\mathcal{S}$.

For space reasons, more details can be found in the long version, available at http://www.diku.dk/~stovring/sgdt.pdf

II. The $\mathcal{S}$ Topos

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

$$\begin{tikzcd}
X(1) \arrow[d, "f_1"'] & X(2) \arrow[l] \arrow[d, "f_2"'] & X(3) \arrow[l] \arrow[d, "f_3"'] & \ldots \arrow[l] \\
Y(1) & Y(2) \arrow[l] & Y(3) \arrow[l] & \ldots \arrow[l]
\end{tikzcd}$$

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, $\mathcal{S}$ 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 $Y^X(n)$ is the set of tuples $(f_1, \ldots, f_n)$ commuting with the restriction maps, and the restriction maps of Y^X 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 $\chi_A \colon X \to \Omega$ 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 $\mathcal{S}$ is the constant set of natural numbers.

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

There is a natural transformation $\mathrm{next}_X \colon X \to \blacktriangleright X$ whose 1st component is the unique map into $\{\star\}$ and whose $(n+1)$st component is r_n.

Since $\blacktriangleright$ preserves finite limits, there is always a morphism

$$J : \blacktriangleright(X \to Y) \to (\blacktriangleright X \to \blacktriangleright Y). \tag{1}$$

An operator on predicates: There is a morphism $\triangleright \colon \Omega \to \Omega$ 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 $\blacktriangleright$ functor, since there is a pullback diagram

$$\begin{tikzcd}
\triangleright m \arrow[r] \arrow[d] \arrow[dr, phantom, "\lrcorner", very near start] & \blacktriangleright A \arrow[d, "\blacktriangleright m"] \\
X \arrow[r, "\mathrm{next}_X"] & \blacktriangleright X
\end{tikzcd}$$

for any subobject $m \colon A \to X$.

Recursive morphisms: We introduce a notion of contractive morphism and show that these have unique fixed points.

Definition II.1. *A morphism $f \colon X \to Y$ is* contractive *if there exists a morphism $g \colon \blacktriangleright X \to Y$ such that $f = g \circ \mathrm{next}_X$. A morphism $f \colon X \times Y \to Z$ is contractive in the first variable if there exists g such that $f = g \circ (\mathrm{next}_X \times \mathrm{id}_Y)$.*

For instance, contractiveness of $\triangleright$ on Ω is witnessed by $\mathrm{succ} \colon \blacktriangleright \Omega \to \Omega$ with $\mathrm{succ}_n(k) = k + 1$.

If $f \colon X \to Y$ 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 $x \colon 1 \to X$ by defining $x_1 = g_1(\star)$ 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 II.2. *There exists a natural family of morphisms $\mathrm{fix}_X \colon (\blacktriangleright X \to X) \to X$, indexed by the collection of all objects X, which computes unique fixed points in the sense that if $f \colon X \times Y \to X$ is contractive in the first variable as witnessed by g, i.e., $f = g \circ (\mathrm{next}_X \times \mathrm{id}_Y)$, then $\mathrm{fix}_X \circ \ulcorner g \urcorner$ is the unique $h \colon Y \to X$ such that $f \circ \langle h, \mathrm{id}_Y \rangle = h$ (here $\ulcorner g \urcorner$ denotes the exponential transpose of g).*

A. Internal logic

We start by calling to mind parts of the Kripke-Joyal forcing semantics for $\mathcal{S}$. For $X \in \mathcal{S}$, $\varphi : X_1 \times \cdots \times X_m \to \Omega$, $n \in \omega$, 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, Example 9.5] (we write $\overline{\alpha}$ for a sequence

$\alpha_1, \ldots, \alpha_m$):

$$
\begin{aligned}
n \models (s = t)\overline{\alpha} &\Leftrightarrow \llbracket s \rrbracket_n(\overline{\alpha}) = \llbracket t \rrbracket_n(\overline{\alpha}) \\
n \models R(t_1, \ldots, t_k)\overline{\alpha} &\Leftrightarrow n \leq \llbracket R \rrbracket_n(\llbracket t_1 \rrbracket_n(\overline{\alpha}), \ldots, \llbracket t_k \rrbracket_n(\overline{\alpha})) \\
n \models (\varphi \wedge \psi)(\overline{\alpha}) &\Leftrightarrow n \models \varphi(\overline{\alpha}) \wedge n \models \psi(\overline{\alpha}) \\
n \models (\varphi \vee \psi)(\overline{\alpha}) &\Leftrightarrow n \models \varphi(\overline{\alpha}) \vee n \models \psi(\overline{\alpha}) \\
n \models (\varphi \rightarrow \psi)(\overline{\alpha}) &\Leftrightarrow \forall k \leq n.\, k \models \varphi(\overline{\alpha}|_k) \rightarrow k \models \psi(\overline{\alpha}|_k) \\
n \models (\exists x{:}X.\varphi)(\overline{\alpha}) &\Leftrightarrow \exists \alpha \in \llbracket X \rrbracket(n).\, n \models \varphi(\overline{\alpha}, \alpha) \\
n \models (\forall x{:}X.\varphi)(\overline{\alpha}) &\Leftrightarrow \forall k \leq n, \alpha \in \llbracket X \rrbracket(k).\, k \models \varphi(\overline{\alpha}|_k, \alpha)
\end{aligned}
$$

Proposition II.3. *$\triangleright$ is the unique morphism on Ω satisfying $1 \models \triangleright\varphi(\alpha)$ and $n+1 \models \triangleright\varphi(\alpha) \Leftrightarrow n \models \varphi(\alpha|_n)$. Moreover, $\forall x, y\colon X.\, \triangleright(x = y) \leftrightarrow \mathrm{next}_X(x) = \mathrm{next}_X(y)$ holds in $\mathcal{S}$.*

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

Definition II.4. *An object X in $\mathcal{S}$ 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 next_X is internally surjective in $\mathcal{S}$, i.e., iff $\forall y : \blacktriangleright X. \exists x : X. \mathrm{next}_X(x) = y$ holds in $\mathcal{S}$. 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 II.5. *In the internal logic of $\mathcal{S}$ we have:*

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

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

Definition II.6. *The predicate* Contr *on Y^X is defined in the internal logic by*

$$\mathrm{Contr}(f) \overset{\mathrm{def}}{\Longleftrightarrow} \forall x, x' : X. \triangleright(x = x') \rightarrow f(x) = f(x').$$

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

Theorem II.7 (Internal Banach Fixed-Point Theorem)**.** *The following holds in $\mathcal{S}$:*

$$(\exists x : X. \top) \wedge \mathrm{Contr}(f) \rightarrow \exists! x : X.\, f(x) = x.$$

For a morphism $f : X \rightarrow Y$, corresponding to a global element of Y^X, we have that if f is contractive (in the external sense of Definition II.1), then $\mathrm{Contr}(f)$ holds in the logic of $\mathcal{S}$. 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 functors (see Section II-B), 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 II.8. *The following holds in $\mathcal{S}$:*

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

Recursive relations: As an example application of Theorem II.7, we consider the definition of recursive predicates. Let $\varphi(r) : \Omega^X$ be a predicate on X in the internal logic of $\mathcal{S}$ as presented above (over non-dependent types, but possibly using $\triangleright$) with free variable r, also of type Ω^X. Note that Ω^X is inhabited by a global element. If r only occurs under a $\triangleright$ in φ, then φ defines an internally contractive map $\varphi\colon \Omega^X \rightarrow \Omega^X$ (proved by external induction on φ). Therefore, by Theorem II.7, $\exists!\, r\colon \Omega^X. \varphi(r) = r$ holds in $\mathcal{S}$. 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 $\mathcal{S}$, 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 φ.

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 II.9. *Suppose $R \subseteq X \times X$ is some relation on a set X. We can include it into $\mathcal{S}$ by using the functor $\Delta\colon \mathbf{Set} \rightarrow \mathcal{S}$, obtaining $\Delta R \subseteq \Delta X \times \Delta X$. Consider the recursive relation*

$$R^\omega(x, y) \overset{\mathrm{def}}{\Longleftrightarrow} (x = y) \vee \exists z. (\Delta R(x, z) \wedge \triangleright R^\omega(z, y)) \,.$$

Now, $1 \models R^\omega(x, y)$ always holds and $n + 1 \models R^\omega(x, y)$ iff $(x, y) \in \cup_{0 \leq i < n} R^i$ or there exists z such that $R^n(x, z)$ holds. If R is a rewrite relation then $n+1 \models R^\omega(x, y)$ 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 II.9 is used in Section III.

B. Recursive domain equations

In this section we present a simplified version of our results on solutions to recursive domain equations in $\mathcal{S}$ sufficient for the example of Section III. The full results on recursive domain equations can be found in Section IV.

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

Definition II.10. *A strong endofunctor on $\mathcal{S}$ is* locally contractive *if each $F_{X,Y}$ is contractive.*

This notion readily generalizes to mixed-variance endofunctors on $\mathcal{S}$. For example, $\blacktriangleright$ 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 $\blacktriangleright$ and simple type constructors in which X occurs only negatively and Y only positively and both only under $\blacktriangleright$ gives rise to a locally contractive functor.

Theorem II.11. *Let $F \colon \mathcal{S}^{\mathrm{op}} \times \mathcal{S} \to \mathcal{S}$ 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 II.11 we sketch the construction to illustrate the use of the locally contractive functors. We consider first the covariant case.

Lemma II.12. *Let $F \colon \mathcal{S} \to \mathcal{S}$ be locally contractive and say that $f \colon X \to Y, g \colon Y \to X$ 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 $p = F! \colon F^2 1 \to F1$ and e as the composition

$$\begin{tikzcd} F1 \arrow[r, "\delta"] & F1 \times F1 \arrow[r, "\mathrm{st}"] & 1 \times F^2 1 \arrow[r, phantom, "\cong"] & F^2 1 \end{tikzcd}$$

where δ is the diagonal and st is the strength corresponding to $F_{-,-}$ [20]. By Lemma II.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 $F^{n+1}1(n)$. For our formal proof we derived a limit / colimit coincidence of the sequence of injection / projection pairs

$$\begin{tikzcd} F1 \arrow[r, shift right, "e"'] & F^2 1 \arrow[l, shift right, "p"'] \arrow[r, shift right, "Fe"'] & F^3 1 \arrow[l, shift right, "Fp"'] \arrow[r, shift right, "F^2 e"'] & F^4 1 \arrow[l, shift right, "F^2 p"'] & \ldots \end{tikzcd}$$

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

Thus, $\mathcal{S}$ 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.

III. 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 $\mathcal{S}$. 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 III-A to III-D are in the internal logic of $\mathcal{S}$. In Section III-E we investigate what these results mean externally.

A. Language

The types and terms of $\mathsf{F}_{\mu,\mathsf{ref}}$ are as follows:

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

$$t ::= x \mid l \mid () \mid (t_1, t_2) \mid \mathsf{fst}\,t \mid \mathsf{snd}\,t \mid \mathsf{fold}\,t \mid \mathsf{unfold}\,t \mid \\ \Lambda\alpha.t \mid t\,[\tau] \mid \bar{\lambda}x.t \mid t_1\,t_2 \mid \mathsf{ref}\,t \mid !t \mid t_1 := t_2$$

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

More explicitly, the sets OType and OTerm of possibly open types and terms are defined by induction according to the grammars above (using that toposes model W-types [22]), and then by quotienting with respect to α-equivalence.

The set OValue of syntactic values is an inductively defined subset of OTerm:

$$v ::= x \mid l \mid () \mid (v_1, v_2) \mid \mathsf{fold}\,v \mid \Lambda\alpha.t \mid \bar{\lambda}x.t$$

Let Term and Value be the subsets of closed terms and closed values, respectively. Let 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 $\mathrm{Config} = \mathrm{Term} \times \mathrm{Store}$.

The typing judgements have the form $\Xi \mid \Gamma \vdash t : \tau$ where Ξ is a context of type variables and Γ is a context of term variables. The typing rules are standard and can be found in the long version of the paper. 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.

B. Operational semantics

We define a standard one-step relation $\mathrm{step}\colon \mathcal{P}(\mathrm{Config} \times \mathrm{Config})$ 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 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 II.7 to define the predicate $\mathrm{eval}\colon \mathcal{P}(\mathrm{Term} \times \mathrm{Store} \times \mathcal{P}(\mathrm{Value} \times \mathrm{Store}))$,

$$\begin{aligned}
&\mathrm{eval}(t, s, Q) \\
&\quad \overset{\mathrm{def}}{\iff} (t \in \mathrm{Value} \ \wedge\ Q(t, s)) \vee \\
&\qquad (\exists t_1\colon \mathrm{Term}, s_1\colon \mathrm{Store}. \\
&\qquad\quad \mathrm{step}((t, s), (t_1, s_1)) \ \wedge\ \triangleright \mathrm{eval}(t_1, s_1, Q))
\end{aligned}$$

Intuitively, the predicate Q is a post-condition, and $\mathrm{eval}(t, s, Q)$ 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 III-E 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 Löb 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 $\mathrm{step}((t, s), (t, s))$ holds, then $\mathrm{eval}(t, s, Q)$ holds for any Q. The Löb rule also proves the following results, which are used to show the fundamental theorem below.

Proposition III.1. *Let $Q, Q' \in \mathcal{P}(\mathrm{Value} \times \mathrm{Store})$ such that $Q \subseteq Q'$. Then for all t and s we have that $\mathrm{eval}(t, s, Q)$ implies $\mathrm{eval}(t, s, Q')$.*

Proposition III.2. *For all stores s, all terms t, all evaluation contexts E such that $E[t]$ is closed, and all predicates $Q \in \mathcal{P}(\mathrm{Value} \times \mathrm{Store})$, we have that $\mathrm{eval}(E[t], s, Q)$ holds iff $\mathrm{eval}(t, s,\ \lambda(v_1, s_1).\, \mathrm{eval}(E[v_1], s_1, Q))$ holds.*

C. Definition of Kripke worlds

The main idea behind our interpretation of types is as in [4, 7]: Since $\mathsf{F}_{\mu,\mathsf{ref}}$ 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

$$\widehat{\mathcal{T}} = \mu X.\ \blacktriangleright((N \to_{\mathrm{fin}} X) \to_{\mathrm{mon}} \mathcal{P}(\mathrm{Value}))\ .$$

Here $N \to_{\mathrm{fin}} X$ is the set $\sum_{A\colon \mathcal{P}_{\mathrm{fin}}(N)} X^A$ where $\mathcal{P}_{\mathrm{fin}}(N) = \{A \subset N \mid \exists m \forall n \in A.\, n < m\}$ ordered by graph inclusion and $\to_{\mathrm{mon}}$ is the set of monotonic functions realized as a subset type on the function space.

One way to see that $\widehat{\mathcal{T}}$ is well-defined is to check that it is a functorial type in the sense of Section IV. Alternatively, observe that the corresponding functor is of the form $F = \blacktriangleright \circ G$. Here G is strong because its action on morphisms can be defined as a term $Y^X \to GY^{GX}$ in the internal logic. Now, since $\blacktriangleright$ is locally contractive so is F. Hence by Theorem II.11, F has a unique fixed point $\widehat{\mathcal{T}}$, with an isomorphism $i\colon \widehat{\mathcal{T}} \to F(\widehat{\mathcal{T}})$. We define

$$\mathcal{W} = N \to_{\mathrm{fin}} \widehat{\mathcal{T}}\ , \qquad \mathcal{T} = \mathcal{W} \to_{\mathrm{mon}} \mathcal{P}(\mathrm{Value})\ ,$$

and $\mathcal{T}^{\mathrm{c}} = \mathcal{W} \to \mathcal{P}(\mathrm{Term})$. Notice that $\widehat{\mathcal{T}}$ is isomorphic to $\blacktriangleright \mathcal{T}$. We now define $\mathrm{app}\colon \widehat{\mathcal{T}} \to \mathcal{T}$ and $\mathrm{lam}\colon \mathcal{T} \to \widehat{\mathcal{T}}$ as follows. First, app is the isomorphism i composed with the operator $d\colon\ \blacktriangleright \mathcal{T} \to \mathcal{T}$ given by

$$d(f) = \lambda w.\lambda v.\mathrm{succ}(J(J(f)(\mathrm{next} w))(\mathrm{next} v)),$$

where J is the map in (1) and $\mathrm{succ}\colon\ \blacktriangleright \Omega \to \Omega$ is as defined on page 3. (This is a general way of lifting algebras for $\blacktriangleright$ to function spaces.) Here one needs to check that d is well-defined, i.e., preserves monotonicity. Second, $\mathrm{lam}\colon \mathcal{T} \to \widehat{\mathcal{T}}$ is defined by $\mathrm{lam} = i^{-1} \circ \mathrm{next}_{\mathcal{T}}$.

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

Lemma III.3. $\mathrm{app} \circ \mathrm{lam} = \triangleright : \mathcal{T} \to \mathcal{T}$.

D. Interpretation of types

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

$$[\![\cdot]\!] : \prod_{\tau \in \mathrm{OType}} \mathrm{TEnv}(\tau) \to \mathcal{T}\ .$$

We show some cases of the definition here; the complete definition can be found in the long version.

$$\begin{aligned}
\llbracket \alpha \rrbracket \varphi &= \varphi(\alpha) \\
\llbracket \tau_1 \times \tau_2 \rrbracket \varphi &= \lambda w.\, \{(v_1,\, v_2) \mid v_1 \in \llbracket \tau_1 \rrbracket \varphi(w) \wedge v_2 \in \llbracket \tau_2 \rrbracket \varphi(w)\} \\
\llbracket \mathsf{ref}\, \tau \rrbracket \varphi &= \lambda w.\, \{\, l \mid l \in \mathrm{dom}(w) \wedge \forall w_1 \geq w. \\
&\qquad\qquad \mathrm{app}(w(l))(w_1) = \triangleright(\llbracket \tau \rrbracket \varphi)(w_1) \,\} \\
\llbracket \forall \alpha.\tau \rrbracket \varphi &= \lambda w.\, \{\, \Lambda\alpha.t \mid \forall \nu \in \mathcal{T}.\, \forall w_1 \geq w. \\
&\qquad\qquad t \in \mathit{comp}(\llbracket \tau \rrbracket \varphi[\alpha \mapsto \nu])(w_1) \,\}
\end{aligned}$$

$$\begin{aligned}
\llbracket \mu\alpha.\tau \rrbracket \varphi &= \mathit{fix}\, (\lambda\nu.\, \lambda w.\, \{\, \mathsf{fold}\, v \mid \triangleright(v \in \llbracket \tau \rrbracket \varphi[\alpha \mapsto \nu]\, (w))\}) \\
\llbracket \tau_1 \to \tau_2 \rrbracket \varphi &= \lambda w.\, \{\, \overline{\lambda} x.t \mid \forall w_1 \geq w. \forall v \in \llbracket \tau_1 \rrbracket \varphi(w_1). \\
&\qquad\qquad t[v/x] \in \mathit{comp}(\llbracket \tau_2 \rrbracket \varphi)(w_1) \,\}
\end{aligned}$$

Here the operations $\mathit{comp} : \mathcal{T} \to \mathcal{T}^{\mathrm{c}}$ and $\mathit{states} : \mathcal{W} \to \mathcal{P}(\mathrm{Store})$ are given by

$$\begin{aligned}
\mathit{comp}(\nu)(w) = \{\, t \mid \forall s \in \mathit{states}(w).\, &\mathit{eval}(t, s, \\
&\lambda(v_1, s_1).\, \exists w_1 \geq w. \\
&\quad v_1 \in \nu(w_1) \;\wedge\; s_1 \in \mathit{states}(w_1)) \,\} \\
\mathit{states}(w) = \{\, s \mid \mathrm{dom}(s) = \mathrm{dom}(w) \;\wedge& \\
\forall l \in \mathrm{dom}(w).\, s(l) \in \mathrm{app}&(w(l))(w) \,\}.
\end{aligned}$$

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 $\llbracket \mu\alpha.\tau \rrbracket$, the use of $\triangleright$ ensures that the fixed point is well-defined according to Theorem II.7. As for the definition of $\llbracket \mathsf{ref}\, \tau \rrbracket$, the $\triangleright$ is needed because we have $\triangleright$ instead of the identity in Lemma III.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 III.4 (Fundamental theorem)**.** *If $\vdash t : \tau$, then for all $w \in \mathcal{W}$ we have $t \in \mathit{comp}(\llbracket \tau \rrbracket \emptyset)(w)$.*

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 the long version of the article. To illustrate the use of $\triangleright$, we outline the case of reference lookup: $\vdash\, !t : \tau$. Here the essential proof obligation is that $v \in \llbracket \mathsf{ref}\, \tau \rrbracket \emptyset(w)$ implies $!v \in \mathit{comp}(\llbracket \tau \rrbracket \emptyset)(w)$. To show this, we unfold the definition of comp. Let $s \in \mathit{states}(w)$ be given; we must show

$$\begin{aligned}
\mathit{eval}(!v, s,\, \lambda(v_1, s_1).\, &\exists w_1 \geq w. \\
&v_1 \in \llbracket \tau \rrbracket \emptyset(w_1) \;\wedge\; s_1 \in \mathit{states}(w_1))\,. \qquad (2)
\end{aligned}$$

By the assumption that $v \in \llbracket \mathsf{ref}\, \tau \rrbracket \emptyset(w)$, we know that $v = l$ for some location l such that $l \in \mathrm{dom}(w)$ and $\mathrm{app}(w(l))(w_1) = \triangleright(\llbracket \tau \rrbracket \emptyset)(w_1)$ for all $w_1 \geq w$. Since $s \in \mathit{states}(w)$, we know that $l \in \mathrm{dom}(s) = \mathrm{dom}(w)$ and $s(l) \in \mathrm{app}(w(l))(w)$. We therefore have $\mathrm{step}((!v, s), (s(l), s))$. Hence, by unfolding the definition of eval in (2) and using the rules from Proposition II.5, it remains to show that $\exists w_1 \geq w.\, \triangleright(s(l) \in \llbracket \tau \rrbracket \emptyset(w_1)) \;\wedge\; \triangleright(s \in \mathit{states}(w_1))$. We choose $w_1 = w$. First, $s \in \mathit{states}(w)$ and hence $\triangleright(s \in \mathit{states}(w))$. Second, $s(l) \in \mathrm{app}(w(l))(w) = \triangleright(\llbracket \tau \rrbracket \emptyset)(w)$, which means exactly that $\triangleright(s(l) \in \llbracket \tau \rrbracket \emptyset(w))$. ■

E. The view from the outside

We now return to the standard universe of sets and give external interpretations of the internal results above. One basic ingredient is the fact that the constant-presheaf functor $\Delta : \mathbf{Set} \to \mathcal{S}$ 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 [22].

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

Proposition III.5. $\mathrm{OType} \cong \Delta(\mathrm{OType}')$*, and similarly for* OTerm*,* Value*,* Store*, and* Config*. Moreover, under these isomorphisms* step *corresponds to* $\Delta\mathrm{step}'$ *as a subobject of* $\mathrm{Config} \times \mathrm{Config}$*.*

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 $n \models \mathrm{step}((t', s'), (t', s'))$ holds iff (t, s) actually steps to (t', s') in the standard operational semantics. We next consider the eval predicate:

Proposition III.6. $n \models \mathrm{eval}(t, s, Q)$ *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 II-A, 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 $\mathcal{P}(\mathrm{Value})(n)$ 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.

F. 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 formulation of parts of [6], which involve recursively defined relations on recursively defined types.

As mentioned in the Introduction, in [4] we solved the recursive equation for $\mathcal{T}$ in the category *CBUlt* of ultrametric spaces. In *loc. cit.* we then moved back into the usual universe of sets and defined the model in the standard, explicit step-indexed style. Here instead we observe that the relevant part of *CBUlt* is a full subcategory of $\mathcal{S}$

(Section V), solve the recursive equation in $\mathcal{S}$, and then *stay* within $\mathcal{S}$ to give a simpler model that does not refer to step indices. In particular, the proof of the fundamental theorem is much simpler when done in $\mathcal{S}$.

IV. Dependent Types

Since $\mathcal{S}$ is a topos it models not only higher-order logic over simple type theory, but also over dependent type theory. The aim of this section is to provide the semantic foundation for extending the dependent type theory with type constructors corresponding to $\blacktriangleright$ and guarded recursive types, although we postpone a detailed syntactic formulation of such a type theory to a later paper.

Recall that dependent types in context are interpreted in slice categories,[4] in particular a type $\Gamma \vdash A$ is interpreted as an object of $\mathcal{S}/[\![\Gamma]\!]$. To extend the interpretation of dependent type theory with a type constructor corresponding to $\blacktriangleright$, we must therefore extend the definition of $\blacktriangleright$ to slice categories.

Generalising $\blacktriangleright$ to slices: We first define $\blacktriangleright_I \colon \mathcal{S}/I \to \mathcal{S}/I$. It acts on objects by mapping $p_Y \colon Y \to I$ to the pullback

$$\begin{tikzcd}
\blacktriangleright_I Y \arrow[r] \arrow[d, "p_{\blacktriangleright_I Y}"'] \arrow[dr, phantom, "\lrcorner", very near start] & \blacktriangleright Y \arrow[d, "\blacktriangleright p_Y"] \\
I \arrow[r, "\mathrm{next}"] & \blacktriangleright I
\end{tikzcd}$$

Define $\mathrm{next}^I_{p_Y} \colon p_Y \to \blacktriangleright_I p_Y$ as the morphism into the pullback corresponding to the pair (p_Y, next_Y).

The definition above allows us to consider $\blacktriangleright$ as a type constructor on dependent types, interpreting $[\![\Gamma \vdash \blacktriangleright A]\!] = \blacktriangleright_{[\![\Gamma]\!]}([\![\Gamma \vdash A]\!])$. The following proposition expresses that this interpretation of $\blacktriangleright$ behaves well wrt. substitution.

Proposition IV.1. *For every $u \colon J \to I$ in $\mathcal{S}$ there is a natural isomorphism $u^* \circ \blacktriangleright_I \cong \blacktriangleright_J \circ u^*$. As a consequence, the collection of functors $(\blacktriangleright_I)_{I \in \mathcal{S}}$ define a fibred endofunctor on the codomain fibration. Moreover,* next *defines a fibred natural transformation from the fibred identity on the codomain fibration to $\blacktriangleright$.*

We remark that each $\blacktriangleright_I$ has a left adjoint, but since the family of left adjoints does not commute with reindexing, it does not define a well-behaved dependent type constructor.

Recursive dependent types: Since the slices of $\mathcal{S}$ are cartesian closed, the notions of strong functors and locally contractive functors from Definition II.10 also make sense in slices. Indeed, the notion of local contractiveness can be generalised to functors enriched over $\mathcal{S}$, and one can talk about enriched functors of several variables being locally contractive in particular variables. The next two lemmas will be used to construct families of locally contractive functors.

Lemma IV.2. *The functor $\blacktriangleright_I \colon \mathcal{S}/I \to \mathcal{S}/I$ is strong and locally contractive.*

Lemma IV.3. *Let $F \colon \mathbb{C} \to \mathbb{D}$, $G \colon \mathbb{D} \to \mathbb{E}$ be $\mathcal{S}/I$-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 $\breve{F} \colon (\mathbb{C}^{\mathrm{op}} \times \mathbb{C})^n \to \mathbb{C}^{\mathrm{op}} \times \mathbb{C}$ of a functor $F \colon (\mathbb{C}^{\mathrm{op}} \times \mathbb{C})^n \to \mathbb{C}$ defined as $\breve{F}(\vec{X}, \vec{Y}) = \langle F(\vec{Y}, \vec{X}), F(\vec{X}, \vec{Y}) \rangle$.

Theorem IV.4. *Let $F \colon ((\mathcal{S}/I)^{\mathrm{op}} \times \mathcal{S}/I)^{n+1} \to \mathcal{S}/I$ be locally contractive in the $(n+1)$st variable pair. Then there exists a unique (up to isomorphism) $\operatorname{Fix} F \colon ((\mathcal{S}/I)^{\mathrm{op}} \times \mathcal{S}/I)^n \to \mathcal{S}/I$ such that $F \circ \langle \mathrm{id}, \breve{\operatorname{Fix} F} \rangle \cong \operatorname{Fix} F$. Moreover, if F is locally contractive in all variables, so is $\operatorname{Fix} F$.*

One can prove that the fixed points obtained by Theorem IV.4 are initial dialgebras 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 [24].

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

Proposition IV.5. *If*

$$\begin{tikzcd}
((\mathcal{S}/I)^{\mathrm{op}} \times \mathcal{S}/I)^{n+1} \arrow[r, "F"] \arrow[d, "u^*"'] & \mathcal{S}/I \arrow[d, "u^*"] \\
((\mathcal{S}/J)^{\mathrm{op}} \times \mathcal{S}/J)^{n+1} \arrow[r, "G"] & \mathcal{S}/J
\end{tikzcd}$$

commutes up to isomorphism, so does

$$\begin{tikzcd}
((\mathcal{S}/I)^{\mathrm{op}} \times \mathcal{S}/I)^{n} \arrow[r, "\operatorname{Fix} F"] \arrow[d, "u^*"'] & \mathcal{S}/I \arrow[d, "u^*"] \\
((\mathcal{S}/J)^{\mathrm{op}} \times \mathcal{S}/J)^{n} \arrow[r, "\operatorname{Fix} G"] & \mathcal{S}/J
\end{tikzcd}$$

Functorial 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.

$$\begin{aligned}
A(\vec{X}^-, \vec{X}^+) ::= {} & X_i^+ \mid C \mid A(\vec{X}^-, \vec{X}^+) \times A(\vec{X}^-, \vec{X}^+) \mid \\
& A(\vec{X}^+, \vec{X}^-) \to A(\vec{X}^-, \vec{X}^+) \mid \\
& \textstyle\prod_{i \colon I} A(\vec{X}^-, \vec{X}^+) \mid \sum_{i \colon I} A(\vec{X}^-, \vec{X}^+) \mid \\
& \{a \colon A(\vec{X}^-, \vec{X}^+) \mid \phi_{\vec{X}^-, \vec{X}^+}(a)\} \mid \\
& \blacktriangleright A(\vec{X}^-, \vec{X}^+) \mid \mu X. A((\vec{X}^-, X), (\vec{X}^+, X))
\end{aligned}$$

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 Γ, satisfying the grammar above and two further requirements stated below, induces a strong functor

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

[4] We follow the practise of ignoring coherence issues related to the interpretation of substitution in codomain fibrations since there are various ways to avoid these issues, e.g. [17].

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

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

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 ϕ 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 $A((\vec{X}^-, Y^-), (\vec{X}^+, Y^+))$ is locally contractive in Y^- and 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 $\blacktriangleright$. If the functorial type A is contractive, then, by Lemma IV.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 $\widehat{\mathcal{T}}$ from the previous section is defined by a functorial type.

V. Relation to metric spaces

Let CBUlt be the category of complete bounded ultrametric spaces and non-expansive maps. In [4–7, 26] 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 $\mathit{BiCBUlt}$ be the full subcategory of CBUlt of bisected spaces, and let BiUlt be the category of all bisected ultrametric spaces (necessarily bounded).

Let $t\mathcal{S}$ be the full subcategory of $\mathcal{S}$ on the total objects.

Proposition V.1. *There is an adjunction between BiUlt and $\mathcal{S}$, which restricts to an equivalence between $t\mathcal{S}$ and $\mathit{BiCBUlt}$, as in the diagram:*

$$\begin{tikzcd}
t\mathcal{S} \arrow[r, shift right] \arrow[d, shift left] \arrow[r, phantom, "\top"] \arrow[d, phantom, "\cong"] & \mathcal{S} \arrow[l, shift right] \arrow[d, shift left] \arrow[d, phantom, "\dashv"] \\
\mathit{BiCBUlt} \arrow[u, shift left] \arrow[r, shift right] \arrow[r, phantom, "\perp"] & \mathit{BiUlt} \arrow[l, shift right] \arrow[u, shift left, "F"]
\end{tikzcd}$$

Proof sketch: The functor $F : \mathit{BiUlt} \to \mathcal{S}$ is defined as follows. A space $(X, d) \in \mathit{BiUlt}$ 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 $t\mathcal{S}$ and that F has a right adjoint that maps into $\mathit{BiCBUlt}$. The right adjoint maps a variable set into a metric space on the limit of the family of variable sets; the metric expresses up to what level elements in the limit agree. The left adjoint from BiUlt to $\mathit{BiCBUlt}$ is then obtained by composition of functors; it is the Cauchy-completion. ■

Proposition V.2. *A morphism in $\mathit{BiCBUlt}$ is contractive in the metric sense iff it is contractive in the internal sense of $\mathcal{S}$.*

The later operator on $\mathcal{S}$ corresponds to multiplying by $\frac{1}{2}$ in ultra-metric spaces, except on the empty space. Specifically, $F(\frac{1}{2}X)$ is isomorphic to $\blacktriangleright(FX)$, for all non-empty X. For ultra-metric spaces, the formulation of existence of solutions to guarded recursive domain equations has to consider the empty space as a special case. Here, in $\mathcal{S}$, we do not have to do so, since $\blacktriangleright$ behaves better than $\frac{1}{2}$ on the empty set.

VI. Conclusion and Future Work

We have shown that the topos $\mathcal{S}$ of trees provides a model of an extension of higher-order logic over dependent type theory with guarded recursive types and terms. Moreover, we have argued that this logic provides the right setting for the synthetic construction of step-indexed models of programming languages and program logics, by constructing a model of the programming language $\mathsf{F}_{\mu,\mathsf{ref}}$ in the logic.

In this paper we have focused solely on guarded recursion. As future work, it would be be interesting to study further the connections between guarded and unguarded recursion in $\mathcal{S}$. 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 conjecture that other models can be constructed as sheaf categories over other well-founded complete Heyting algebras than $\overline{\omega}$, building on the work of Di Gianantonio and Miculan [8]. However, the existence theorem for recursive types would then have to be reproved (e.g., it might involve taking a limit over a larger diagram in case the Heyting algebra is larger than $\overline{\omega}$).

We plan to make a tool for formalized reasoning in the internal logic of $\mathcal{S}$. 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. This work was supported in part by grants from the Danish research council (Birkedal and Møgelberg) and from the Carlsberg Foundation (Støvring).

References

[1] M. Abbott, T. Altenkirch, and N. Ghani. Representing nested inductive types using W-types. In *Proc. of ICALP*, 2004.

[2] A. Ahmed, D. Dreyer, and A. Rossberg. State-dependent representation independence. In *Proc. of POPL*, 2009.

[3] A.W. Appel, P.-A. Melliès, C.D. Richards, and J. Vouillon. A very modal model of a modern, major, general type system. In *Proc. of POPL*, 2007.

[4] L. Birkedal, B. Reus, J. Schwinghammer, K. Støvring, J. Thamsborg, and H. Yang. Step-indexed Kripke models over recursive worlds. In *Proc. of POPL*, 2011.

[5] L. Birkedal, J. Schwinghammer, and K. Støvring. A metric model of lambda calculus with guarded recursion. Presented at FICS, 2010.

[6] L. Birkedal, J. Schwinghammer, and K. Støvring. A step-indexed Kripke model of hidden state via recursive properties on recursively defined metric spaces. In *Proc. of FOSSACS*, 2011.

[7] L. Birkedal, K. Støvring, and J. Thamsborg. Realisability semantics of parametric polymorphism, general references and recursive types. *Math. Struct. Comp. Sci.*, 20(4):655–703, 2010.

[8] P. Di Gianantonio and M. Miculan. Unifying recursive and co-recursive definitions in sheaf categories. In *Proc. of FOSSACS*, 2004.

[9] D. Dreyer, A. Ahmed, and L. Birkedal. Logical step-indexed logical relations. In *Proc. of LICS*, 2009.

[10] D. Dreyer, G. Neis, and L. Birkedal. The impact of higher-order state and control effects on local relational reasoning. In *Proc. of ICFP*, 2010.

[11] D. Dreyer, G. Neis, A. Rossberg, and L. Birkedal. A relational modal logic for higher-order stateful ADTs. In *Proc. of POPL*, 2010.

[12] P. Dybjer. Representing inductively defined sets by wellorderings in Martin-Löf's type theory. *Theor. Comput. Sci.*, 176(1-2):329–335, 1997.

[13] P.J. Freyd. Recursive types reduced to inductive types. In *Proc. of LICS*, 1990.

[14] P.J. Freyd. Algebraically complete categories. In *Proc. of the 1990 Como Category Theory Conference*, 1991.

[15] P.J. Freyd. Remarks on algebraically compact categories. In *Applications of Categories in Computer Science*, volume 177 of *LMS Lecture Note Series*, 1991.

[16] F. Garillot, G. Gonthier, A. Mahboubi, and L. Rideau. Packaging mathematical structures. In *Proc. of TPHOLs*, 2009.

[17] M. Hofmann. On the interpretation of type theory in locally cartesian closed categories. In *Proc. of CSL*, 1994.

[18] J.M.E. Hyland. First steps in synthetic domain theory. In *Proc. of the 1990 Como Category Theory Conference*, 1991.

[19] J.M.E. Hyland, P.T. Johnstone, and A.M. Pitts. Tripos theory. *Math. Proc. of the Cambridge Philosophical Society*, 88:205–232, 1980.

[20] A. Kock. Strong functors and monoidal monads. *Arch. Math.*, 23:113–120, 1972.

[21] J. Lambek and P.J. Scott. *Introduction to higher order categorical logic*. Cambridge University Press, 1986.

[22] I. Moerdijk and E. Palmgren. Wellfounded trees in categories. *Annals of Pure and Appl. Logic*, 104:189–218, 2000.

[23] H. Nakano. A modality for recursion. In *Proc. of LICS*, 2000.

[24] A.M. Pitts. Relational properties of domains. *Inf. Comput.*, 127(2):66–90, 1996.

[25] F. Pottier. A typed store-passing translation for general references. In *Proc. of POPL*, 2011.

[26] J. Schwinghammer, H. Yang, L. Birkedal, F. Pottier, and B. Reus. A semantic foundation for hidden state. In *Proc. of FOSSACS*, 2010.