# Denotational semantics of general store and polymorphism

JONATHAN STERLING, Aarhus University, Denmark
DANIEL GRATZER, Aarhus University, Denmark
LARS BIRKEDAL, Aarhus University, Denmark

We contribute the first denotational semantics of polymorphic dependent type theory extended by an equational theory for general (higher-order) reference types and recursive types, based on a combination of guarded recursion and impredicative polymorphism; because our model is based on *recursively defined semantic worlds*, it is compatible with polymorphism and relational reasoning about stateful abstract datatypes. We then extend our language with modal constructs for proof-relevant relational reasoning based on the *logical relations as types* principle, in which equivalences between imperative abstract datatypes can be established synthetically. Finally we decompose our store model as a general construction that extends an arbitrary polymorphic call-by-push-value adjunction with higher-order store, improving on Levy's possible worlds model construction; what is new in relation to prior typed denotational models of higher-order store is that our Kripke worlds need not be syntactically definable, and are thus compatible with relational reasoning in the heap. Our work combines recent advances in the operational semantics of state with the purely denotational viewpoint of *synthetic guarded domain theory*.

CCS Concepts: • **Theory of computation → Type theory**; Abstraction; **Categorical semantics**; **Denotational semantics**.

## 1 INTRODUCTION

The combination of parametric polymorphism and general reference types in denotational semantics is notoriously difficult, though neither feature presents any serious difficulties on its own.

(1) Parametric polymorphism can be modelled in a *complete internal category* à la Hyland [1988]; for instance, the category of partial equivalence relations has "large" products over the category of assemblies, which contains the assembly of all partial equivalence relations.
(2) General reference types can be modeled via a Kripke world-indexed state monad, where the worlds assign *syntactic* types to locations in the heap, as in the possible worlds model of thunk storage by Levy [2002, 2003a].

The two techniques described above are not easily combined: in order to support parametric reasoning in the presence of reference types, it is necessary at a minimum for the Kripke worlds to assign *semantic* types to locations rather than only syntactic types. But a semantic type should itself be a family of (sets, predomains, *etc.*) indexed in Kripke worlds; thus one attempts to solve a domain equation of that defines a preorder $\mathbb{W}$ of Kripke worlds simultaneously with the category of functors [Oles 1986; Reynolds 1981] from $\mathbb{W}$ to the category of predomains:

$$\mathbb{W} \cong \mathsf{Loc} \rightharpoonup_{fin.} \mathsf{SemType} \qquad \mathsf{SemType} \cong [\mathbb{W}, \mathsf{Predomain}] \tag{$*$}$$

There are already a few problems with the "domain equation" presented above. For one, the collection of predomains is not itself a predomain, but it could be replaced by the domain of finitary projections of some universal domain [Coquand et al. 1994]. The more fundamental problem is that it remains unclear how to interpret type connectives on SemType; for instance, Birkedal et al. [2010, §5] provide an explicit counterexample to demonstrate that a naïve interpretation of the reference type cannot coexist with recursive types in the presence of semantic worlds.

Authors' addresses: Jonathan Sterling, jmsterli@cs.au.dk, Aarhus University, 34 Aabogade, Aarhus, Denmark, 8200; Daniel Gratzer, gratzer@cs.au.dk, Aarhus University, 34 Aabogade, Aarhus, Denmark, 8200; Lars Birkedal, birkedal@cs.au.dk, Aarhus University, 34 Aabogade, Aarhus, Denmark, 8200.

### 1.1 Step-indexing and guarded domain theory

The difficulties outlined above have been side-stepped by the introduction of *step-indexing* [Ahmed 2004; Appel et al. 2007] and its semantic counterpart *metric/guarded domain theory* [America and Rutten 1987; Birkedal et al. 2011b; Rutten and Turi 1992], which proceed by stratifying recursive definitions in their finite approximations. This stratification was axiomatized by Birkedal et al. [2011a] in ***synthetic guarded domain theory*** / SGDT via the ***later modality*** $\blacktriangleright : \mathscr{S} \longrightarrow \mathscr{S}$ which comes equipped with a point $\mathsf{next} : \mathrm{id}_{\mathscr{S}} \longrightarrow \blacktriangleright$; the standard model of SGDT is the topos of trees $\mathscr{S} = \mathbf{Pr}\,\omega$ where the later modality is defined like so:

$$(\blacktriangleright A)_n = \varprojlim_{k<n} A_k \qquad \mathsf{next}^n_A x = k \mapsto x_{|k}$$

In the setting of SGDT, any recursive definition has a unique fixed point if its recursive variables are guarded by an application of $\blacktriangleright$. In categorical terms, $\mathscr{S}$ is *algebraically compact* with respect to *locally contractive* endofunctors. Letting $\mathbb{U}$ be a type universe in a model $\mathscr{S}$ of synthetic guarded domain theory, it is easy to solve the following approximate domain equation:

$$\mathbb{W} \cong \mathsf{Loc} \rightharpoonup_{fin.} \blacktriangleright\mathsf{SemType} \qquad \mathsf{SemType} \cong [\mathbb{W}, \mathbb{U}]$$

The preorder $\mathbb{W}$ of Kripke worlds and the collection of semantic types SemType both exist as objects of $\mathscr{S}$; it is even possible to define a suitable connective $\mathrm{ref} : \mathsf{SemType} \longrightarrow \mathsf{SemType}$. We may also define an indexed type of heaps $\mathbb{H} : \mathbb{W} \to \mathbb{U}$; unfortunately it is quite unclear to define the *indexed state monad* for $\mathbb{H}$ as a connective on SemType. For instance, the standard definition of the indexed state monad [Levy 2004; Plotkin and Power 2002; Power 2011] runs into obvious size problems given that $\mathsf{SemType} \notin \mathbb{U}$ and thus $\mathbb{W} \notin \mathbb{U}$:

$$\mathsf{T}\,Aw = \prod_{w' \geq w} \mathbb{H}w' \to \sum_{w'' \geq w'} \mathbb{H}w'' \times Aw'' \tag{$*$}$$

Indeed, the definition above cannot be executed in the standard model $\mathscr{S} = \mathbf{Pr}\,\omega$ when $\mathbb{U}$ is the Hofmann–Streicher lifting of a Grothendieck universe from **Set**. This problem is side-stepped in *operational* models, where $\mathbb{U}$ is replaced by a set of predicates on syntactical expressions of the programming language being modeled and T is defined by a guarded version of weakest preconditions as in Birkedal et al. [2011a]:

$$\begin{aligned} \mathsf{T}\,Aw &= \{u \in \mathsf{Val} \mid \forall w' \geq w, h \in \mathbb{H}w'.\mathsf{wp}(w', h, u())\{(w'', h', v) \mapsto v \in Aw''\}\} \\ \mathsf{wp}(w, h, e)\{\Phi\} &= (e \in \mathsf{Val} \wedge \Phi(w, h, e)) \vee \exists w', h', e'.(h; e) \mapsto (h'; e') \wedge \blacktriangleright\mathsf{wp}(w', h')\{\Phi\} \end{aligned}$$

Operational methods worked in $\mathscr{S}$ *only because* the set of predicates is a complete lattice: we may compute the join and intersection of arbitrarily large families of predicates. This observation will ultimately form the basis for our own non-operational solution; rather than giving up on denotations, we will work in a *different* model of synthetic guarded domain theory that contains an ***impredicative universe*** $\mathbb{U}$, *i.e.* one closed under universal and (thus) existential types à la System F; then we can define the indexed state monad as follows without encountering size problems:

$$\mathsf{T}\,Aw = \forall_{w' \geq w} \mathbb{H}w' \to \exists_{w'' \geq w'} \mathbb{H}w'' \times Aw''$$

This approach raises the question: *does there in fact exist a model of synthetic guarded domain theory with an impredicative universe?* We answer in the affirmative, constructing a model in presheaves on a well-founded order internal to a ***realizability topos*** [van Oosten 2008]. Our model of SGDT is thus an instance of the relative-topos-theoretic generalization of synthetic guarded domain theory introduced by Palombi and Sterling [2022].

## 1.2 This paper: impredicative guarded dependent type theory with reference types

Our denotational model of higher-order store was immediately suited for generalization to dependent types; this scalability is one of many advantages our abstract category-theoretic methods. Justified by this model, we have defined Impredicative Guarded Dependent Type Theory (**iGDTT**) and its extension with general reference types ($\mathbf{iGDTT}^{\mathsf{ref}}$) and their equational theory.

$\mathbf{iGDTT}^{\mathsf{ref}}$ is the first language to soundly combine "full-spectrum" dependently typed programming and equational reasoning with both higher-order store and recursive types. The necessity of *semantics* for higher-order store with dependent types is not hypothetical: both Idris 2 and Lean 4 are dependently typed, higher-order functional programming languages [Brady 2021; De Moura and Ullrich 2021] that feature a Haskell-style `IO` monad with general `IORef` types [Jones 2001], but until now these features have had no semantics. In fact, our investigations have revealed a subtle problem in the `IO` monads of both Idris and Lean: as higher-order store is inherently impredicative, it does not make sense to close multiple nested universes under the `IO` monad [Coquand 1986].

Many real-world examples require one to go *beyond* simple equational reasoning; to this end, we defined an extension $\mathbf{iGDTT}^{\mathsf{ref}}_{\mathrm{LRAT}}$ with constructs for synthetic, proof-relevant relational reasoning for data abstraction and weak bisimulation based on the ***logical relations as types*** (LRAT) principle of Sterling and Harper [2021], which axiomatizes a generalization of the parametricity translation of dependent type theory [Bernardy et al. 2012; Bernardy and Moulin 2012].

## 1.3 Discussion of related work

*1.3.1 Operational semantics of higher-order store.* The most thoroughly developed methods for giving semantics to higher-order store are based on *operational semantics* [Ahmed 2004]; accompanying the operational semantics of higher-order store is a wealth of powerful program logics for modular reasoning about higher-order effectful and even concurrent programs based on higher-order separation logic [Bizjak and Birkedal 2018; Jung et al. 2018; Svendsen and Birkedal 2014].

We are motivated to pursue denotations for three reasons. First, denotational methods are amenable to the use of general theorems from other mathematical fields to solve difficult domain problems, whereas operational methods tend to require one to solve every problem "by hand" without relying on standard lemmas. Secondly, a good denotational semantics tends to simplify reasoning about programs, as the exponents of the DeepSpec project have argued [Xia et al. 2019]. Finally and most profoundly, there is an emerging need to program directly with *actual spaces*, as in differentiable and probabilistic programming languages [Abadi and Plotkin 2019; Vákár et al. 2019].

Although operational and denotational semantics are different in both purpose and technique, there is nonetheless a rich interplay between the two traditions. First of all, the idea of step-indexing and its approximate equational theory lies at the heart of our denotational semantics; second, the wealth of results in operationally based models and program logics for higher-order store suggest several areas for future work in our denotational semantics. For instance, it would be interesting to rebase a program logic such as Iris atop our synthetic denotational model; likewise, operationally based works such as those of Ahmed et al. [2009]; Dreyer et al. [2010] suggest many improvements to our notion of *semantic world* to support richer kinds of correspondence between programs.

*1.3.2 Denotational semantics of state.* There is a rich tradition of denotational semantics of state, ranging from a single reference cell [Moggi 1991] to first-order store [Plotkin and Power 2002] and storage of pointers [Kammar et al. 2017], and even higher-order store with syntactic worlds [Levy 2004]. None of these approaches is compatible with relational reasoning for polymorphic/abstract types, as they all rely on the semantic heap being classified by syntactically definable types.

*Untyped metric semantics and approximate locations.* A somewhat different *untyped* approach to the denotational semantics of higher-order store with recursive types and polymorphism was pioneered by Birkedal et al. [2010] in which one uses metric/guarded domain theory to define a *universal domain* D, and then develops a model of System $\mathbf{F}_{\mu}^{\mathsf{ref}}$ in predicates on D. Because predicates form a complete lattice, it is possible to interpret the state operations as we have discussed in Section 1.1. An important aspect of this class of models is the presence of *approximate locations*, which *op. cit.* have shown to be non-optional.[1]

*Comparison.* Our own model resembles a synthetic version of that of Birkedal et al. [2010], but there are some important differences. Our model is considerably more abstract and more general than that of *op. cit.*; whereas Birkedal et al. must solve a very complex metric domain equation to construct a universal domain, we may use the realizability topos generated by *any* partial combinatory algebra, and thus we do not depend on any specific choice of universal domain. Moreover, we argue that ours is a *typed* model: we depend only on the combination of guarded recursion and an impredicative universe, and although the principle source of such structures is realizability on untyped partial combinatory algebras [Lietz and Streicher 2002], we do not depend on any of the details of realizability. Because of the generality and abstractness of our model construction, scaling up to full dependent type theory has offered no resistance.

*1.3.3 Higher-order store in Impredicative Hoare Type Theory.* Realizability models of impredicative type theory have been used before to give a model of so-called Impredicative Hoare Type Theory (**iHTT**), an extension of dependent type theory with a monadic "Hoare" type for stateful computations with higher-order store [Svendsen et al. 2011]. However, in contrast to our monadic type in $\mathbf{iGDTT}^{\mathsf{ref}}$, **iHTT** does not support equational reasoning about computations: all elements of a Hoare type are equated and one can only reason via the types. Moreover, the Hoare type in *op. cit.* only supports untyped locations with substructural reference capabilities that change as the heap evolves (the so-called "strong update"); thus non-Hoare types are not indexed in worlds.

*1.3.4 Linear dependent type theory.* Another approach to the integration of state into dependent type theory is contributed by Krishnaswami et al. [2015], who develop an adjoint linear–non-linear dependent type theory $\mathbf{LNL}_{\mathrm{D}}$ with a realizability model in partial equivalence relations. Like **iHTT**, the theory of *op. cit.* uses capabilities for references with strong update; unlike **iHTT**, the account of store in $\mathbf{LNL}_{\mathrm{D}}$ enjoys a rich equational theory. What is missing from $\mathbf{LNL}_{\mathrm{D}}$ is any account of general recursion, which normally arises from higher-order store via backpatching or Landin's Knot; indeed, $\mathbf{LNL}_{\mathrm{D}}$ carefully avoids the general recursive aspects of higher-order store via linearity. One of the main advances of our own paper over both **iHTT** and $\mathbf{LNL}_{\mathrm{D}}$ is to model a dependently typed equational theory for *full* store with general recursion.

*1.3.5 Effectful dependent type theory.* We have not attempted any non-trivial interaction between computational effects and the dependent type structure of our language, in contrast to the work of Pédrot and Tabareau [2019] on dependent call-by-push-value: our $\mathbf{iGDTT}^{\mathsf{ref}}$ language can be thought of as a purely functional programming language extended with a monad for stateful programming with Haskell-style `IORefs`.

Our polymorphic call-by-push-value (cbpv) decomposition of the store model is mainly inspired by the work of Vákár [2017] on dependently typed generalizations of Levy's adjunction models of cbpv [Levy 2003b]. Our work is also closely related to the syntactical account of dependent cbpv by Pédrot and Tabareau [2019], but it is not directly comparable as *op. cit.* have focused mainly on

[1] Indeed, even in our own model the presence of approximate locations can be detected using Thamsborg's observation on the correspondence between metric and ordinary domain theory [Thamsborg 2010, Ch. 9].

a syntactic *weaning* translation that tracks the Eilenberg–Moore algebra models of cbpv, whereas our own model is not of this form. We expect, however, that our model would be related in some form to the *forcing* translation of *op. cit*.

### 1.4 Structure and contributions of this paper

Our contributions are as follows:

- In **Section** 2 we introduce ***impredicative guarded dependent type theory*** (**iGDTT**) as a user-friendly metalanguage for the denotational semantics of languages involving polymorphism, general reference types and recursive types. In **Section** 2.4 as a case study, we construct a simple denotational model of Monadic System $\mathbf{F}_{\mu}^{\mathsf{ref}}$, a language with general reference types, polymorphic types, and recursive types in **iGDTT**. Finally in **Section** 2.5 we describe our Coq library for **iGDTT** in which we have formalized the higher-order state monad and reference types.
- In **Section** 3 we describe $\mathbf{iGDTT}^{\mathsf{ref}}$, an extension of **iGDTT** with general reference types and a monad for higher-order store. $\mathbf{iGDTT}^{\mathsf{ref}}$ is thus a full-spectrum dependently typed programming language with support for higher-order effectful programming. To illustrate the combination of higher-order store with dependent types, we define and prove the correctness of an implementation of factorial defined via ***Landin's knot*** / backpatching.
- In **Section** 4 we describe $\mathbf{iGDTT}^{\mathsf{ref}}_{\mathrm{LRAT}}$, an extension of $\mathbf{iGDTT}^{\mathsf{ref}}$ with constructs for synthetic proof-relevant relational reasoning based on the ***logical relations as types*** principle [Sterling and Harper 2021]. $\mathbf{iGDTT}^{\mathsf{ref}}_{\mathrm{LRAT}}$ can be used to succinctly exhibit bisimulations between higher-order stateful computations and abstract data types, which we demonstrate in two case studies involving imperative counter implementations (Sections 4.3 and 4.4).
- In **Section** 5, we describe general results for constructing models of **iGDTT** and $\mathbf{iGDTT}^{\mathsf{ref}}_{\mathrm{LRAT}}$, with concrete instantiations given by a combination of realizability and internal presheaves. The results of this section justify the consistency of $\mathbf{iGDTT}^{\mathsf{ref}}_{\mathrm{LRAT}}$ as a language for relational reasoning about higher-order stateful computations.
- In **Section** 6 we decompose our model of higher-order store as a ***generic model construction*** that applies to any polymorphic adjunction model of call-by-push-value. Thus our model of higher-order store can be combined modularly with other computational effects, extending the result of Levy [2003b] for *monomorphic* higher-order store and *syntactic* worlds to the more difficult case of polymorphism and semantic worlds.
- In **Section** 7 we conclude with some reflections on directions for future work.

## 2 IMPREDICATIVE GUARDED DEPENDENT TYPE THEORY

In this section we describe an extension of *guarded dependent type theory* with an impredicative universe; guarded dependent type theory is a dependently typed interface to synthetic guarded domain theory. The purpose of this ***impredicative guarded dependent type theory*** (**iGDTT**) is to serve as a metalanguage for the denotational semantics of programming languages involving general reference types, just as ordinary guarded dependent type theory can be used as a metalanguage for denotational semantics of programming languages with recursive functions [Paviotti et al. 2015] and recursive types [Møgelberg and Paviotti 2016].

### 2.1 Universe structure: quantifiers and reflection

The core **iGDTT** language is modelled off of Martin-Löf type theory with a pair of impredicative base universes $(\mathbb{P} \subseteq \mathbb{U}) \in \mathbb{V}_0 \in \mathbb{V}_1 \in \ldots$, as in the version of the calculus of inductive constructions with *impredicative Set.*[2] The universes $\mathbb{U}, \mathbb{V}_i$ are closed under dependent products, dependent sums, finite enumerations $[n]$, inductive types (W-types), and extensional equality types with equality reflection. We assert that $\mathbb{P}$ is both proof-irrelevant and univalent, and moreover closed under extensional equality types. Proof-irrelevance means that for any $P : \mathbb{P}$ and $p, q : P$ we have $p = q$; univalence means that if $P \leftrightarrow Q$ then $P = Q$.[3]

What makes $\mathbb{U}, \mathbb{P}$ impredicative is that we assert an additional connective for *universal types* with abstraction, application, $\beta$-, and $\eta$-laws.

$$\textsc{impredicativity}\quad \frac{\mathbb{S} \in \{\mathbb{U}, \mathbb{P}\} \qquad A : \mathbb{V}_i \qquad x : A \vdash Bx : \mathbb{S}}{\forall_{x:A}\, Bx : \mathbb{S}}$$

The universal type automatically gives rise to an impredicative encoding of *existential* types. The naïve encoding $\exists_{x:A}\, Bx \triangleq^{*} \forall_{C:\mathbb{S}} \big(\forall_{x:A} (Bx \to C)\big) \to C$ does not in fact have the correct universal property as its $\eta$-law holds only up to parametricity, but we may use the method of Awodey et al. [2018] to define a correct version of the existential type. It will be simplest to do so in two steps: first define the *reflection* $\|-\|_{\mathbb{S}} : \mathbb{V}_i \to \mathbb{S}$, and then apply this reflection to the dependent sum.

Theorem 2.1. *The inclusion* $\mathbb{S} \hookrightarrow \mathbb{V}_i$ *has a left adjoint* $\|-\|_{\mathbb{S}} : \mathbb{V}_i \to \mathbb{S}$.

Proof. For reasons of space, we give only the definition of $\|-\|_{\mathbb{S}}$. The reflection is constructed in two steps; first we define the "wild" reflection $|-|_{\mathbb{S}} : \mathbb{V}_i \to \mathbb{S}$ by an impredicative encoding, which is unfortunately too unconstrained to have the universal property of the left adjoint:

$$\begin{aligned} &|-|_{\mathbb{S}} : \mathbb{V}_i \to \mathbb{S} \\ &|A|_{\mathbb{S}} \triangleq \forall_{C:\mathbb{S}} (A \to C) \to C \end{aligned}$$

We therefore constrain $|A|_{\mathbb{S}}$ by a naturality condition, encoded as a structure $\mathsf{ok}_A : |A|_{\mathbb{S}} \to \mathbb{S}$ defined using universal and equality types like so:

$$\begin{aligned} &\mathsf{ok}_A : |A|_{\mathbb{S}} \to \mathbb{S} \\ &\mathsf{ok}_A \alpha \triangleq \forall_{C,D:\mathbb{S}} \forall_{f:C \to D} \forall_{h:A \to C}\, \alpha\, D\, (f \circ h) = f(\alpha\, C\, h) \end{aligned} \qquad \begin{aligned} &\|-\|_{\mathbb{S}} : \mathbb{V}_i \to \mathbb{S} \\ &\|A\|_{\mathbb{S}} \triangleq \textstyle\sum_{\alpha:|A|_{\mathbb{S}}} \mathsf{ok}_A \alpha \end{aligned}$$

This completes the construction of the reflection. □

We will write $\mathsf{pack} : A \to \|A\|_{\mathbb{S}}$ for the unit of the reflection; via the universal property of the adjunction, maps into types classified by $\mathbb{S}$ can be defined by pattern matching, *e.g.* $\mathbf{let}\, \mathsf{pack}\, x = u\, \mathbf{in}\, v$; by virtue of the constraint $\mathsf{ok}_A$, these destructuring expressions satisfy a desirable $\eta$-law. With the reflection in hand, it is possible to give a correct encoding of the existential type:

$$\frac{A : \mathbb{V}_i \qquad x : A \vdash Bx : \mathbb{S}}{\exists_{x:A}\, Bx \triangleq \big\|\sum_{x:A} Bx\big\|_{\mathbb{S}}}$$

The naïve impredicative encoding would have worked for $\mathbb{P}$ because it is already proof-irrelevant; but for uniformity, we present the reflections for $\mathbb{P}, \mathbb{U}$ simultaneously.

## 2.2 The later modality and delayed substitutions

The remainder of **iGDTT**—the guarded fragment—is the same in prior presentations [Bizjak et al. 2016]; to summarize, we have a type operator $\blacktriangleright$ called the ***later modality*** equipped with a fixed-point operator. We defer a proper exposition of guarded type theory to Bizjak et al. [2016];

[2] We mean that every element of $\mathbb{P}$ is also classified by $\mathbb{U}$, but we do not assert that $\mathbb{P}$ is classified by $\mathbb{U}$.

[3] This is also called *propositional extensionality.*

intuitively, however, the type $\blacktriangleright A$ contains elements of $A$ which only become available one 'step' in the future. We have chosen to present $\blacktriangleright$ using the ***delayed substitutions*** $\boxed{\xi \rightsquigarrow \Xi}$ of *op. cit.*:

$$\frac{\xi \rightsquigarrow \Xi \qquad \Xi \vdash A \ \mathit{type}}{\blacktriangleright[\xi].A \ \mathit{type}} \qquad \frac{\xi \rightsquigarrow \Xi \qquad \Xi \vdash a : A}{\mathsf{next}[\xi].a : \blacktriangleright[\xi].A} \qquad \frac{}{\cdot \rightsquigarrow \cdot} \qquad \frac{\xi \rightsquigarrow \Xi \qquad a : \blacktriangleright[\xi].A}{(\xi, x \leftarrow a) \rightsquigarrow \Xi, x : A}$$

The delayed substitution attached to the introduction and formation rules allows an element of $\blacktriangleright A$ to strip away the $\blacktriangleright$-modalities from a list of terms, thereby rendering $\blacktriangleright$ an applicative functor in the sense of McBride and Paterson [2008]:

$$f \circledast a \triangleq \mathsf{next}[x \leftarrow f, y \leftarrow a].xy$$

All universes $\mathbb{X} \in \{\mathbb{U}, \mathbb{P}, \mathbb{V}_i\}$ are closed under $\blacktriangleright$. We have equational laws governing both delayed substitutions and the $\blacktriangleright$/next constructors; we will not belabor them here, referring instead to *op. cit.* for a precise presentation. In the case of empty delayed substitutions $\xi = \cdot$, we will write $\blacktriangleright A$ and next $a$ for $\blacktriangleright[\cdot].A$ and $\mathsf{next}[\cdot].a$ respectively. A guarded fixed point combinator is included:

$$\frac{x : \blacktriangleright A \vdash fx : A}{\mathsf{gfix}\, x.fx : A} \qquad \frac{x : \blacktriangleright A \vdash fx : A}{\mathsf{gfix}\, x.fx = f\,(\mathsf{next}\,(\mathsf{gfix}\, x.fx)) : A}$$

### 2.3 Guarded domains and the lift monad

By construction, all types in **iGDTT** support a guarded fixed-point operator $(\blacktriangleright A \to A) \to A$; without an algebra structure $\blacktriangleright A \to A$, however, such a fixed point operator is insufficient to interpret general recursion. We will define a ***guarded domain*** to be a type equipped with exactly such an algebra structure below.

*Definition 2.2.* We define a ***guarded domain*** to be a type $A$ together with a function $\vartheta_A : \blacktriangleright A \to A$, *i.e.* an algebra for the later modality viewed as an endofunctor.

Given a guarded domain $A$, we define the ***delay map*** $\delta_A : A \to A$ to be the composite $\vartheta_A \circ \mathsf{next}$. A guarded domain can be equipped with a fixed point combinator $\mu_A : (A \to A) \to A$ satisfying $\mu_A f = \delta_A\,(f\,(\mu_A f))$. In particular, we define $\mu_A f \triangleq \mathsf{gfix}\, x.\, \vartheta_A\,(\mathsf{next}_A[z \leftarrow x].fz)$.

*Example 2.3.* Each universe $\mathbb{X} \in \{\mathbb{P}, \mathbb{U}, \mathbb{V}_i\}$ carries the structure of a guarded domain, as we may define $\vartheta_{\mathbb{X}} A \triangleq \blacktriangleright[X \leftarrow A].X$. Note that we have $\delta_{\mathbb{X}} A = \blacktriangleright A$.

Because the universe is itself a guarded domain, the same fixed point combinators can be used to interpret *both* recursive programs *and* recursive types as pointed out by Birkedal and Møgelberg [2013]; this is a significant improvement over ordinary (synthetic) domain theory, where more complex notions of algebraic compactness are required to lift recursion to the level of types.

*Construction 2.4 (Guarded lift monad).* We will write $\mathbf{GDom}(\mathbb{X})$ for the universe of guarded domains in a universe $\mathbb{X}$; the forgetful functor $\mathsf{J} : \mathbf{GDom}(\mathbb{X}) \longrightarrow \mathbb{X}$ has a left adjoint $\mathsf{L} \dashv \mathsf{J}$ that freely *lifts* a type to a guarded domain, which we may compute by taking a guarded fixed point:

$$\mathsf{L}\,A = A + \blacktriangleright \mathsf{L}\,A \qquad\qquad \vartheta_{\mathsf{L}\,A} \triangleq \mathsf{inr}$$

Unfolding definitions, we have solved the guarded domain equation $\mathsf{L}\,A = A + \blacktriangleright \mathsf{L}\,A$; we will write $\eta : A \to \mathsf{L}\,A$ for the left injection. When it causes no confusion, we will leave the forgetful functor $\mathsf{J}$ implicit; thus we refer to $\mathsf{L} : \mathbb{X} \longrightarrow \mathbb{X}$ as the ***guarded lift monad***.

## 2.4 Case study: denotational semantics of Monadic System $\mathbf{F}_\mu^{\mathsf{ref}}$

Already we have enough machinery to explore a simple possible worlds model of System $\mathbf{F}_\mu^{\mathsf{ref}}$, a polymorphic language with general reference types. Even in this simple case, the tension between polymorphism and general reference types is evident and we require both the impredicative and guarded-recursive features of **iGDTT** to construct the ***semantic worlds*** that underly the model. We emphasize that the entire construction takes place internally to **iGDTT**.

For the moment, we only sketch the interpretation of types in our model. We will return to this point in Section 5, when we use a similar definition of semantic worlds to construct a model for a version of full **iGDTT** extended with general references.

*2.4.1 Recursively defined semantic worlds and heaps.* We define a function World : $\mathbb{V}_0 \longrightarrow \mathbb{V}_0$ taking a type $A : \mathbb{V}_0$ to the preorder of finite maps $\mathbb{N} \rightharpoonup_{\mathit{fin.}} A$, where the order is given by graph inclusion. Next we define $\mathcal{T} : \mathbb{V}_0$ to be the guarded fixed point of the operator that sends $A : \blacktriangleright\mathbb{V}_0$ to the type of functors $[\mathsf{World}(\blacktriangleright[z \leftarrow A]z), \mathbb{U}]$; thus we have $\mathcal{T} = [\mathsf{World}(\blacktriangleright\mathcal{T}), \mathbb{U}]$. Finally we define $\mathbb{W}$ to be the preorder $\mathsf{World}(\blacktriangleright\mathcal{T})$. Thus we have solved the domain equation $\mathbb{W} = \mathbb{N} \rightharpoonup_{\mathit{fin.}} \blacktriangleright[\mathbb{W}, \mathbb{U}]$ and we have $\mathcal{T} = \mathsf{ob}_{[\mathbb{W},\mathbb{U}]}$. The object $\mathcal{T}$ can be seen to be a guarded domain, setting $\vartheta_{\mathcal{T}} A \triangleq \lambda w.\blacktriangleright[X \leftarrow A].Xw$.

We extend the above to a functor heap : $\mathbb{W}^\circ \longrightarrow [\mathbb{W}, \mathbb{U}]$ as follows, writing $\mathbb{W}^\circ$ for the opposite of the preorder $\mathbb{W}$, by defining heap $ww' \triangleq \prod_{i \in |w|} \vartheta_{\mathcal{T}}(wi)\, w'$. Here we have used $|w|$ to denote the support of the finite mapping $w$. We will write $\mathbb{H}_w$ for heap $w\, w$ and $\mathbb{H} : \mathbb{V}_0$ for the dependent sum $\sum_{w:\mathsf{ob}_{\mathbb{W}}} \mathbb{H}_w$.

*2.4.2 Higher-order state monad and reference types.* We now define a strong monad $\mathsf{T}$ on $[\mathbb{W}, \mathbb{U}]$, a variation on the standard state monad suitable for computations involving state and general recursion. Recalling that $\mathsf{L} : \mathbb{U} \longrightarrow \mathbb{U}$ is the guarded lift monad, we define $\mathsf{T}$ below:

$$\begin{array}{l} \mathsf{T} : [\mathbb{W}, \mathbb{U}] \longrightarrow [\mathbb{W}, \mathbb{U}] \\ (\mathsf{T}\,A)w \triangleq \forall_{w' \geq w}\, \mathbb{H}_{w'} \longrightarrow \mathsf{L}\, \exists_{w'' \geq w'}\, \mathbb{H}_{w''} \times Aw'' \end{array}$$

Theorem 2.5. *$\mathsf{T}$ is a strong monad and $\mathsf{T}\,Aw$ is a guarded domain for each $A : \mathcal{T}$ and $w : \mathbb{W}$.*

Next we define a semantic reference type connective ref : $\mathcal{T} \longrightarrow \mathcal{T}$. For each $A : \mathcal{T}$, the type ref $A$ essentially picks out those locations in the current world which contain elements of type $A$:

$$(\mathsf{ref}\,A)w \triangleq \{l \in |w| \mid \blacktriangleright[B \leftarrow wl].A = B\}$$

*2.4.3 Synthetic model of Monadic System $\mathbf{F}_\mu^{\mathsf{ref}}$.* We may now implement a denotational semantics of Monadic System $\mathbf{F}_\mu^{\mathsf{ref}}$ in **iGDTT**; the direct-style call-by-value version of System $\mathbf{F}_\mu^{\mathsf{ref}}$can then be interpreted separately à la Moggi in the standard way. We specify the domains of interpretation for each form of judgment below:

$$[\![\Xi \vdash]\!] : \mathbb{V} \quad [\![\Xi \mid \Gamma \vdash]\!], [\![\Xi \vdash \tau\ \mathit{type}]\!] : [\![\Xi]\!] \to \mathcal{T} \quad [\![\Xi \mid \Gamma \vdash e : \tau]\!] : \forall_{\xi:[\![\Xi\vdash]\!]} [\![\Xi \mid \Gamma \vdash]\!]\xi \to [\![\Xi \vdash \tau\ \mathit{type}]\!]\xi$$

Individual type connectives are interpreted below:

$$\begin{array}{l} [\![\Xi \vdash \forall\alpha.\tau\ \mathit{type}]\!]\xi w \triangleq \forall_{X:\mathcal{T}} [\![\Xi \vdash \tau\ \mathit{type}]\!](\xi, X)w \\ [\![\Xi \vdash \exists\alpha.\tau\ \mathit{type}]\!]\xi w \triangleq \exists_{X:\mathcal{T}} [\![\Xi \vdash \tau\ \mathit{type}]\!](\xi, X)w \\ [\![\Xi \vdash \mu\alpha.\tau\ \mathit{type}]\!]\xi \triangleq \mu_{\mathcal{T}}(\lambda X.[\![\Xi, \alpha \vdash \tau\ \mathit{type}]\!](\xi, X)) \\ [\![\Xi \vdash \mathsf{ref}\,\tau\ \mathit{type}]\!]\xi \triangleq \mathsf{ref}\,([\![\Xi \vdash \tau\ \mathit{type}]\!]\xi) \\ [\![\Xi \vdash \mathsf{T}\,\tau\ \mathit{type}]\!]\xi \triangleq \mathsf{T}\,([\![\Xi \vdash \tau\ \mathit{type}]\!]\xi) \end{array}$$

Function and product types are interpreted using the cartesian closure of $[\mathbb{W}, \mathbb{U}]$; note that functor categories with relatively large domain are not typically cartesian closed, but in our case the impredicativity of $\mathbb{U} \in \mathbb{V}$ ensures cartesian closure. Next we interpret the state operations as families of natural transformations in $[\mathbb{W}, \mathbb{U}]$:

$$\begin{array}{l} \llbracket \Xi \mid \Gamma \vdash \mathsf{get}_\tau l : \mathsf{T}\,\tau \rrbracket \xi w \gamma w' h \triangleq \\ \quad \mathbf{let}\ i \triangleq \llbracket \Xi \mid \Gamma \vdash l : \mathsf{ref}\,\tau \rrbracket \xi w' \gamma_{|w'}; \\ \quad x \leftarrow \vartheta\,(\mathsf{next}[z \leftarrow hi].\eta z); \\ \quad \eta\,(\mathsf{pack}\,(w', (h, x))) \end{array} \qquad \begin{array}{l} \llbracket \Xi \mid \Gamma \vdash \mathsf{set}_\tau\, l\, e : \mathsf{T}\,1 \rrbracket \xi w \gamma w' h \triangleq \\ \quad \mathbf{let}\ i \triangleq \llbracket \Xi \mid \Gamma \vdash l : \mathsf{ref}\,\tau \rrbracket \xi w' \gamma_{|w'}; \\ \quad \mathbf{let}\ x \triangleq \llbracket \Xi \mid \Gamma \vdash e : \tau \rrbracket \xi w' \gamma_{|w'} \\ \quad \eta\,(\mathsf{pack}\,(w', (h[i \mapsto \mathsf{next}\,x], ()))) \end{array}$$

$$\begin{array}{l} \llbracket \Xi \mid \Gamma \vdash \mathsf{new}_\tau\, e : \mathsf{T}\,(\mathsf{ref}\,\tau) \rrbracket \xi w \gamma w' h \triangleq \\ \quad \mathbf{let}\ i = \mathsf{fresh}\,|w'|; \\ \quad \mathbf{let}\ w'' \triangleq w' \cup \{i \mapsto \mathsf{next}\,(\llbracket \Xi \vdash \tau\ \mathit{type} \rrbracket)\}; \\ \quad \mathbf{let}\ x \triangleq \llbracket \Xi \mid \Gamma \vdash e : \tau \rrbracket \xi w'' \gamma_{|w''}; \\ \quad \eta\,(\mathsf{pack}\,(w'', (h_{|w''} \cup \{l \mapsto \mathsf{next}\,x\}, i))) \end{array}$$

Above in the interpretation of the $\mathsf{new}_\tau$ operator, we have assumed a deterministic "allocator" $\mathsf{fresh} : \mathsf{ob}_{\mathbb{W}} \longrightarrow \mathbb{N}$ that chooses an unused address for a new reference cell. Note that the naturality conditions of the interpretation do not depend in any way on the behavior of fresh.

*2.4.4 Abstract steps in the equational theory of higher-order store.* Notice that our interpretation of the getter must invoke the $\blacktriangleright$-algebra structure on $\mathsf{T}\,A$ as the heap associates to each location $i$ an element of $\vartheta_{\mathcal{T}}(wi)$. For this reason, equations governing *reads* to the heap do not hold on the nose but rather hold up to an abstract "step".

For example, given $l : \mathsf{ref}\,A$ we might expect the equation $\llbracket \mathsf{set}_A\, l\, a; \mathsf{get}_A l \rrbracket = \llbracket \mathsf{set}_A\, l\, a; \mathsf{ret}\, a \rrbracket$ to hold, but the call to $\vartheta$ in the former denotation will prevent them from agreeing in our model. This behavior is familiar already from the work of Escardó [1999]; Paviotti et al. [2015] on denotational semantics of general recursion in guarded/metric domain theory: the guarded interpretation of recursion forces a more intensional notion of equality that counts abstract steps. Thus in order to properly formulate the equations that *do* hold, we must account for these abstract steps. To this end, we extend Monadic System $\mathbf{F}_\mu^{\mathsf{ref}}$ by a new primitive effect $\mathsf{step} : \mathsf{T}\,1$ that takes an abstract step:

$$\llbracket \Xi \mid \Gamma \vdash \mathsf{step} : \mathsf{T}\,1 \rrbracket \xi w \gamma w' h \triangleq \delta\,(\eta\,(\mathsf{pack}\,(w', (h, ()))))$$

With the above in hand, we may state and prove the expected equations for state:

Theorem 2.6. *Our denotational semantics of Monadic System $\mathbf{F}_\mu^{\mathsf{ref}}$ validates the following equations:*

$$\frac{\Xi \mid \Gamma \vdash l : \mathsf{ref}\,\tau \qquad \Xi \mid \Gamma \vdash u : \tau}{\Xi \mid \Gamma \vdash \mathsf{set}_\tau\, l\, u; \mathsf{get}_\tau l \equiv \mathsf{step}; \mathsf{set}_\tau\, l\, u; \mathsf{ret}\, u : \mathsf{T}\,\tau} \qquad \frac{\Xi \mid \Gamma \vdash l : \mathsf{ref}\,\tau}{\Xi \mid \Gamma \vdash (x \leftarrow \mathsf{get}_\tau l; \mathsf{set}_\tau\, l\, x) \equiv \mathsf{step} : \mathsf{T}\,1}$$

$$\frac{\Xi \mid \Gamma \vdash u, v : \tau}{\Xi \mid \Gamma \vdash (x \leftarrow \mathsf{new}_\tau\, u; \mathsf{set}_\tau\, x\, v; \mathsf{ret}\, x) \equiv \mathsf{new}_A\, v : \mathsf{T}\,(\mathsf{ref}\,\tau)} \qquad \frac{\Xi \mid \Gamma \vdash l : \mathsf{ref}\,\tau \qquad \Xi \mid \Gamma \vdash u, v : \tau}{\Xi \mid \Gamma \vdash \mathsf{set}_\tau\, l\, u; \mathsf{set}_\tau\, l\, v \equiv \mathsf{set}_\tau\, l\, v : \mathsf{T}\,1}$$

Proof. Immediate upon unfolding the denotational semantics of the given equations. □

## 2.5 Formal case study: presheaf model of higher-order store in Coq

Although the semantics of **iGDTT** are admittedly quite sophisticated (see Section 5), one of its advantages is that it is quite straightforward to extend existing proof assistants such as Coq [Coq Development Team 2016] and Agda [Norell 2009] with the axioms of **iGDTT**. Thus a practitioner of programming language semantics can define "naïve" synthetic models of effects such as higher-order store in a proof assistant without needing to know the category theory that was required to invent and justify **iGDTT**.

To validate the use of **iGDTT** for formal semantics of programming languages, we have postulated the axioms of **iGDTT** in a Coq library and used it to formalize the construction of the indexed state monad T defined in Section 2.4.2 as well as the reference type connective. In our formalization, we decompose T as the horizontal composition of several adjunctions and thus obtain the monad laws for free by taking T to be the monad of the composite adjunction.

*2.5.1 Axiomatizing the impredicative universe.* We take Coq's `Set` universe to be the impredicative universe $\mathbb{U}$ of **iGDTT**; it is possible to execute Coq with the flag `-impredicative-set`, but we have found it simpler to implement the impredicativity by means of a local pragma as follows:

```
#[bypass_check(universes = yes)]
Definition All {A : Type} {B : A -> Set} : Set :=
  forall x : A, B x.
```

For **iGDTT**'s predicative universes $\mathbb{V}_i$ we use Coq's predicative universes `Type@{i}`. We formalize the left adjoint to the inclusion `Set` ↪ `Type@{i}`, verifying its universal property using the argument of Awodey et al. [2018].

*2.5.2 Axiomatizing guarded recursion.* Next we axiomatize the later modality in Coq. Coq's implicit universe polymorphism ensures that the following axioms land not only in `Type` but in `Set` as well:

```
Axiom later : Type -> Type.
Notation "▶ A" := (later A) (at level 60).
Axiom next : forall A, A -> ▶ A.
Axiom loeb : forall A (f : ▶ A -> A), A.
Axiom loeb_unfold : forall A (f : ▶ A -> A), loeb f = f (next (loeb f)).
```

We do not include the general delayed substitutions in our axiomatization, because they are too difficult to formalize; for our case study, the following special case of delayed substitutions suffices:

```
Axiom dlater : ▶ Type -> Type.
Axiom dlater_next_eq : forall A, ▶ A = dlater (next A).
```

*2.5.3 Formalizing guarded interaction trees.* Building on the above and several other axioms of guarded recursion that we omit for lack of room, we formalize a notion of guarded algebraic effect based on *containers* [Abbott et al. 2005]. Given a container $\mathbb{E} = (S : \mathbb{U}, P : S \longrightarrow \mathbb{U})$, we may view each $s : S$ as the name of an effect operation for which $Ps$ is the continuation arity. Writing $\mathcal{P}_{\mathbb{E}}X = \sum_{s:S} \prod_{p:Ps} X$ for the polynomial endofunctor presented by $\mathbb{E}$, we may consider for each $A$ the free $\mathcal{P}_{\mathbb{E}}$ algebra generated by $A$ and the (co)-free $\mathcal{P}_{\mathbb{E}}$ coalgebra generated by $A$; the former is the inductive type of finite (terminating) computations of a value of type $A$ that use the effects specified by $\mathbb{E}$, whereas the latter is the *coinductive* type of possibly infinite computations. The latter are referred to as *interaction trees* by Xia et al. [2019], who have propelled a resurgence in their use for reasoning about code in first-order languages via denotational semantics.

We formalize a *guarded* version of interaction trees, obtained by solving the guarded domain equation $X = A + \mathcal{P}_{\mathbb{E}} \blacktriangleright X$ using `loeb` in `Set`.

```
Record Container := {op : Set; bdry : op -> Set}.
Inductive ITree_F (𝔼 : Container) A T : Set) : Set :=
| Ret (r : A)
| Do (e : 𝔼) (k : bdry e -> T).
Definition ITree (𝔼 : Container) (A : Set) : Set :=
  loeb (fun T => ITree_F (dlater T)).
```

We develop a free-forgetful adjunction for each container $\mathbb{E}$; in particular, we prove that `ITree 𝔼 A` is the free $(\mathcal{P}_{\mathbb{E}} \circ \blacktriangleright)$-algebra generated by $A$. The reason for developing guarded interaction trees is that there is a particular container $\mathbb{E}$ for which `ITree 𝔼` is the *guarded lift monad* $\mathsf{L}$ of Paviotti et al. [2015], which we have discussed in Section 2.3. Nonetheless, the extra generality allows us to modularly add other effects such as failure or printing, *etc.*

*2.5.4 Recursively defined worlds, presheaves, and the state monad.* We start by defining the category of finite maps of elements of a given type:

```
Definition world : Type -> Category.type := ...
```

Then we define the collections of semantic worlds and semantic types by solving a guarded domain equation, where we write SET for the category induced by the impredicative Set universe.

```
Definition 𝓕 (T : Type) : Type := [world T, SET].
Definition 𝓣 : Type := loeb (fun T => 𝓕 (dlater T)).
Definition 𝕎 : Category.type := world 𝓣.
Definition heap (w : 𝕎) : Type := ...
Definition ℍ : Type := {w : 𝕎 & heap w}.
```

We have also formalized the reference type connective as a map `ref : 𝓣 -> 𝓣`, exactly as in the informal presentation of Section 2.4.2. Our formalization of the state monad differs from our informal presentation in two ways: firstly we have generalized over an arbitrary container $\mathbb{E}$ specifying computational effects (*e.g.* printing, failure, *etc.*), and secondly we found it simpler to decompose it into the following sequence of simpler adjunctions, whose definition involves the impredicative `All` quantifier that we axiomatized earlier:

$$\begin{tikzcd}
{[\mathbb{W}^\circ, \mathtt{Alg}\ \mathbb{E}]} \arrow[r, bend right, "{[\mathbb{W}^\circ, \mathtt{Forget}]}"'] \arrow[r, phantom, "\perp"] & {[\mathbb{W}^\circ, \mathtt{Set}]} \arrow[l, bend right, "{[\mathbb{W}^\circ, \mathtt{ITree}\ \mathbb{E}]}"'] \arrow[r, bend right, "\Delta_{\mathsf{heap}}"'] \arrow[r, phantom, "\perp"] & {[\mathbb{H}, \mathtt{Set}]} \arrow[l, bend right, "\exists_{\mathsf{heap}}"'] \arrow[r, bend right, "\forall_{\mathsf{heap}}"'] \arrow[r, phantom, "\perp"] & {[\mathbb{W}, \mathtt{Set}]} \arrow[l, bend right, "\Delta_{\mathsf{heap}}"']
\end{tikzcd}$$

We did not formalize the state operations, but there is no obstacle to doing so.

# 3 IMPREDICATIVE GUARDED DEPENDENT TYPE THEORY WITH REFERENCE TYPES

In this section, we detail an extension **iGDTT**$^{\text{ref}}$ of **iGDTT** with general reference types and a state monad. **iGDTT**$^{\text{ref}}$ can serve as both an effectful higher-order programming language in its own right, *and* as a dependently typed metalanguage for the denotational semantics of other programming languages involving higher-order store. Later on we will extend **iGDTT**$^{\text{ref}}$ with relational constructs to enable the *verification* of stateful programs inside the type theory. Denotational semantics for these extensions of **iGDTT** are discussed in Section 5.

## 3.1 Adding reference types to iGDTT

In this section, we extend the **iGDTT** language with the following constructs:

(1) a monad $\mathsf{T} : \mathbb{U} \to \mathbb{U}$,
(2) a later algebra structure $\vartheta_{\mathsf{T}A} : \blacktriangleright \mathsf{T}\,A \to \mathsf{T}\,A$ parameterized in $A : \mathbb{U}$ such that:

$$\frac{u : \mathsf{T}\,A \qquad x : A \vdash vx : \mathsf{T}\,B}{\delta_{\mathsf{T}B}(x \leftarrow u; vx) = (x \leftarrow \delta_{\mathsf{T}A} u; vx) = (x \leftarrow u; \delta_{\mathsf{T}B}(vx)) : \mathsf{T}\,B}$$

We will write $\mathsf{step} : \mathsf{T}\,1$ for the generic effect $\delta_{\mathsf{T}1}(\mathsf{ret}\,())$; the rule above ensures that $\mathsf{step}$ commutes with all operations in the monad.

(3) a reference type connective closed under the following rules:

$$\frac{A : \mathbb{U}}{\mathsf{ref}\, A : \mathbb{U}} \qquad \frac{l : \mathsf{ref}\, A}{\mathsf{get}_A l : \mathsf{T}\, A} \qquad \frac{l : \mathsf{ref}\, A \quad u : A}{\mathsf{set}_A\, l\, u : \mathsf{T}\, 1} \qquad \frac{u : A}{\mathsf{new}_A\, u : \mathsf{T}\,(\mathsf{ref}\, A)}$$

$$\frac{l : \mathsf{ref}\, A \quad u : A}{\mathsf{set}_A\, l\, u; \mathsf{get}_A l = \mathsf{step}; \mathsf{set}_A\, l\, u; \mathsf{ret}\, u : \mathsf{T}\, A} \qquad \frac{l : \mathsf{ref}\, A}{(x \leftarrow \mathsf{get}_A l; \mathsf{set}_A\, l\, x) = \mathsf{step} : \mathsf{T}\, 1}$$

$$\frac{u, v : A}{(x \leftarrow \mathsf{new}_A\, u; \mathsf{set}_A\, x\, v; \mathsf{ret}\, x) = \mathsf{new}_A\, v : \mathsf{T}\,(\mathsf{ref}\, A)} \qquad \frac{l : \mathsf{ref}\, A \quad u, v : A}{\mathsf{set}_A\, l\, u; \mathsf{set}_A\, l\, v = \mathsf{set}_A\, l\, v : \mathsf{T}\, 1}$$

### 3.2 Programming with higher-order store and dependent types

*3.2.1 Type dependency in the heap.* Because the universe $\mathbb{U}$ is closed under dependent types (including equality types, *etc.*), there is no obstacle to storing elements of dependent types in the heap. For instance, it is easy to allocate a reference that can only hold even integers; as an example, we consider a program that increments an even integer in place, abbreviating $T \triangleq \{z : \mathbb{Z} \mid \mathsf{isEven}\, z\}$:

$$\begin{aligned} &M : \mathsf{ref}\, T \to \mathsf{T}\, 1 \\ &M \triangleq \lambda l.x \leftarrow \mathsf{get}_T l; \mathsf{set}_T\, l\, (x + 2) \end{aligned}$$

*3.2.2 Recursion via back-patching.* To illustrate higher-order store, we can give a more sophisticated example involving *backpatching* via Landin's knot, writing $\bot : \mathsf{L}\,\alpha$ for the divergent element $\mu x.x$.

$$\begin{aligned} &\mathsf{patch} : \forall_{\alpha:\mathbb{U}}((\alpha \to \mathsf{T}\,\alpha) \to (\alpha \to \mathsf{T}\,\alpha)) \to \mathsf{T}\,(\mathsf{ref}\,(\alpha \to \mathsf{T}\,\alpha)) \\ &\mathsf{patch}\,\alpha\, F \triangleq r \leftarrow \mathsf{new}_{\alpha \to \mathsf{T}\,\alpha}\,(\lambda\_.\bot); \mathsf{set}_{\alpha \to \mathsf{T}\,\alpha}\, r\, (F\,(\lambda x.f \leftarrow \mathsf{get}_{\alpha \to \mathsf{T}\,\alpha} r; f x)); \mathsf{ret}\, r \end{aligned}$$

$$\begin{aligned} &\mathsf{knot} : \forall_{\alpha:\mathbb{U}}((\alpha \to \mathsf{T}\,\alpha) \to (\alpha \to \mathsf{T}\,\alpha)) \to \alpha \to \mathsf{T}\,\alpha \\ &\mathsf{knot}\,\alpha\, F\, x \triangleq r \leftarrow \mathsf{patch}\,\alpha\, F; F\,(\lambda z.f \leftarrow \mathsf{get}_{\alpha \to \mathsf{T}\,\alpha} r; f z)\, x \end{aligned}$$

We can use the backpatching knot to give a somewhat contrived computation of the factorial:

$$\begin{aligned} &\mathsf{fact}' : (\mathbb{N} \to \mathsf{T}\,\mathbb{N}) \to \mathbb{N} \to \mathsf{T}\,\mathbb{N} \\ &\mathsf{fact}'\, f\, 0 \triangleq \mathsf{ret}\, 1 \\ &\mathsf{fact}'\, f\, (n+1) \triangleq m \leftarrow f\, n; \mathsf{ret}\,((n+1) \times m) \end{aligned} \qquad \begin{aligned} &\mathsf{fact} : \mathbb{N} \to \mathsf{T}\,\mathbb{N} \\ &\mathsf{fact} \triangleq \mathsf{knot}\,\mathbb{N}\,\mathsf{fact}' \end{aligned}$$

Lemma 3.1. *Entirely inside of* **iGDTT**$^{\mathsf{ref}}$ *we may prove the following correctness lemma given a reference implementation* $\mathsf{goodFact} : \mathbb{N} \to \mathbb{N}$*:*

$$\forall_{n:\mathbb{N}}\, \mathsf{fact}\, n = (\_ \leftarrow \mathsf{patch}\,\mathbb{N}\,\mathsf{fact}'; \mathsf{step}^n; \mathsf{ret}\,(\mathsf{goodFact}\, n))$$

Proof. We proceed by induction on $n$; the base case is immediate:

$$\begin{aligned} \mathsf{fact}\, 0 &= r \leftarrow \mathsf{patch}\,\mathbb{N}\,\mathsf{fact}'; \mathsf{fact}'\,(\ldots)\, 0 \\ &= \_ \leftarrow \mathsf{patch}\,\mathbb{N}\,\mathsf{fact}'; \mathsf{ret}\, 1 \\ &= \_ \leftarrow \mathsf{patch}\,\mathbb{N}\,\mathsf{fact}'; \mathsf{ret}\,(\mathsf{goodFact}\, 0) \end{aligned}$$

Next we fix $n : \mathbb{N}$ such that $\mathsf{fact}\, n = (\_ \leftarrow \mathsf{patch}\,\mathbb{N}\,\mathsf{fact}'; \mathsf{step}^n; \mathsf{ret}\,(\mathsf{goodFact}\, n))$:

$$\begin{aligned} &\mathsf{fact}\,(n+1) \\ &\quad = r \leftarrow \mathsf{patch}\,\mathbb{N}\,\mathsf{fact}'; \mathsf{fact}'\,(\lambda x.f \leftarrow \mathsf{get}_{\mathbb{N} \to \mathsf{T}\,\mathbb{N}} r; f x)\,(n+1) \\ &\quad = r \leftarrow \mathsf{patch}\,\mathbb{N}\,\mathsf{fact}'; m \leftarrow (f \leftarrow \mathsf{get}_{\mathbb{N} \to \mathsf{T}\,\mathbb{N}} r; f n); \mathsf{ret}\,((n+1) \times m) \\ &\quad = r \leftarrow \mathsf{patch}\,\mathbb{N}\,\mathsf{fact}'; f \leftarrow \mathsf{get}_{\mathbb{N} \to \mathsf{T}\,\mathbb{N}} r; m \leftarrow f n; \mathsf{ret}\,((n+1) \times m) \\ &\quad = r \leftarrow \mathsf{patch}\,\mathbb{N}\,\mathsf{fact}'; \mathsf{step}; m \leftarrow \mathsf{fact}'\,(\lambda x.f \leftarrow \mathsf{get}_{\mathbb{N} \to \mathsf{T}\,\mathbb{N}} r; f x)\, n; \mathsf{ret}\,((n+1) \times m) \end{aligned}$$

$$\begin{aligned}
&= r \leftarrow \mathsf{patch}\,\mathbb{N}\,\mathsf{fact}'; m \leftarrow \mathsf{fact}'\,(\lambda x.f \leftarrow \mathsf{get}_{\mathbb{N}\to\mathsf{T}\,\mathbb{N}} r; f\,x)\,n; \mathsf{step}; \mathsf{ret}\,((n+1)\times m) \\
&= m \leftarrow \mathsf{fact}\,n; \mathsf{step}; \mathsf{ret}\,((n+1)\times m) && \text{by def.} \\
&= \_ \leftarrow \mathsf{patch}\,\mathbb{N}\,\mathsf{fact}'; \mathsf{step}^n; \mathsf{step}; \mathsf{ret}\,((\mathsf{goodFact}\,n+1)\times n) && \text{by i.h.} \\
&= \_ \leftarrow \mathsf{patch}\,\mathbb{N}\,\mathsf{fact}'; \mathsf{step}^{n+1}; \mathsf{ret}\,(\mathsf{goodFact}\,(n+1)) && \square
\end{aligned}$$

*Remark 3.2.* Note that we do not have the equation fact $n$ = ret (goodFact $n$); to understand why this is the case, there are two things to take note of. First of all, the abstract steps taken by each access to the heap are explicitly tracked by the equational theory of **iGDTT**$^{\mathsf{ref}}$ as an instance of the step effect. Secondly, even though the allocations carried out by patch are no longer active upon return, the equational theory of **iGDTT**$^{\mathsf{ref}}$ cannot “garbage collect” them.

## 4 LOGICAL RELATIONS AS TYPES FOR RECURSION AND HIGHER-ORDER STORE

Like any dependent type theory, **iGDTT**$^{\mathsf{ref}}$ is equipped with a “built-in” notion of equality, but it is too fine to serve as a basis for reasoning about the equivalence of effectful programs. For instance, we have already seen that even the most basic programs governing reading and writing to a reference may differ in the number of steps they take. Accordingly, we will construct a coarser notion of equivalence between programs in the form of a logical relation over **iGDTT**$^{\mathsf{ref}}$.

In particular, we discuss an extension of the axioms of **iGDTT**$^{\mathsf{ref}}$ which supports proof-relevant binary relational reasoning in the sense of Sterling and Harper [2021]’s ***logical relations as types*** (LRAT) principle.[4] The result is **iGDTT**$^{\mathsf{ref}}_{\mathsf{LRAT}}$, a new dependent type theory with lightweight features for verifying the equivalence of effectful computations. In addition to capturing the traditional “method of candidates” reasoning, **iGDTT**$^{\mathsf{ref}}_{\mathsf{LRAT}}$ includes a notion of weak bisimulation [Møgelberg and Paviotti 2016; Paviotti 2016]. The inclusion of weak bisimulation is crucial; without it, only programs that read from the heap in lockstep could ever be related.

Finally, we show how our synthetic approach simplifies two standard examples from Birkedal et al. [2010] concerning equivalent implementations of imperative counters.

### 4.1 Logical relations as types: a synthetic approach to relational reasoning

The ***logical relations as types*** principle states that logical relations can be treated *synthetically* as ordinary types in the presence of certain modal operators that exist in categories of logical relations. Thus LRAT is an abstraction of logical relations that avoids the bureaucracy of analytic approaches; the “work” that goes into conventional logical relations arguments has not disappeared, but has been refactored into general results of category theory that are much simpler to establish directly than almost any of their practical consequences in programming languages.

We execute the LRAT extension **iGDTT**$^{\mathsf{ref}}_{\mathsf{LRAT}}$ of **iGDTT**$^{\mathsf{ref}}$ by postulating three propositions $\mathbf{\Phi}_{\mathrm{L}}, \mathbf{\Phi}_{\mathrm{R}}, \mathbf{\Phi} : \mathbb{P}$ representing the left- and right-hand sides of a correspondence, and the disjoint union of both sides; the idea is that these propositions will generate modal operators that allow us to think of *any* type as a correspondence, and then project out different aspects of the correspondence.

$$\frac{}{\mathbf{\Phi}_{\mathrm{L}}, \mathbf{\Phi}_{\mathrm{R}}, \mathbf{\Phi} : \mathbb{P}} \qquad\qquad \frac{\_ : \mathbf{\Phi}_{\mathrm{L}} \qquad \_ : \mathbf{\Phi}_{\mathrm{R}}}{\_ : \bot}$$

Above we have imposed a rule to make $\mathbf{\Phi}_{\mathrm{L}}, \mathbf{\Phi}_{\mathrm{R}}$ disjoint. We additionally add rules to make $\mathbf{\Phi}$ behave as the *coproduct* $\mathbf{\Phi}_{\mathrm{L}} + \mathbf{\Phi}_{\mathrm{R}}$, with a splitting rule not only for propositions but also for types.

[4] The LRAT language is also referred to in other contexts as *synthetic Tait computability*; see Sterling [2021].

$$\frac{\_ : \Phi_{\mathrm{L}} \vdash a_L : A \qquad \_ : \Phi_{\mathrm{R}} \vdash a_R : A}{\_ : \Phi \vdash [\Phi_{\mathrm{L}} \hookrightarrow a_L, \Phi_{\mathrm{R}} \hookrightarrow a_R] : A} \qquad \frac{\_ : \Phi_{\mathrm{L}} \vdash a_L : A \qquad \_ : \Phi_{\mathrm{R}} \vdash a_R : A}{\_ : \Phi_{\mathrm{L}} \vdash [\Phi_{\mathrm{L}} \hookrightarrow a_L, \Phi_{\mathrm{R}} \hookrightarrow a_R] = a_L : A}$$

$$\frac{\_ : \Phi_{\mathrm{L}} \vdash a = b : A \qquad \_ : \Phi_{\mathrm{R}} \vdash a = b : A}{\_ : \Phi \vdash a = b : A}$$

The modalities that we consider in this paper are all *left exact, idempotent, monadic* modalities in the sense of Rijke et al. [2020]. Left exactness means that they preserve finite limits, and idempotence means that the multiplication map $T^2 \longrightarrow T$ is a natural isomorphism. In this section, let $T$ be an arbitrary left exact idempotent monad.

*Definition 4.1 (Modal types).* We will refer to a type $A$ as ***$T$-modal*** when the unit map $\eta_A : A \longrightarrow TA$ is an isomorphism.

*Notation 4.2 (Modal elimination).* Idempotent monadic modalities enjoy an elimination rule with a universal property: for any $T$-modal type $C$, the function $(TA \to C) \longrightarrow (A \to C)$ induced by precomposition with the unit is an isomorphism. We reflect this in our notation as follows:

$$\frac{C \text{ is } T\text{-modal} \qquad u : TA \qquad x : A \vdash fx : C}{(x \leftarrow u; fx) : C} \qquad \frac{\ldots \qquad a : A \qquad x : A \vdash fx : C}{(x \leftarrow \eta a; fx) = fa : C}$$

$$\frac{C \text{ is } T\text{-modal} \qquad x : TA \vdash fx, gx : C \qquad x : A \vdash f\,(\eta x) = g\,(\eta x) : C \qquad u : TA}{fu = gu : C}$$

Let $\phi : \mathbb{P}$ be a proposition.

*Definition 4.3.* A type $A$ is called ***$\phi$-transparent*** when the constant map $A \longrightarrow (\phi \to A)$ is an isomorphism; on the other hand, $A$ is called ***$\phi$-sealed*** when the projection map $\phi \times A \longrightarrow \phi$ is an isomorphism. Equivalently, $A$ is $\phi$-sealed exactly when $\phi$ implies that $A$ is a singleton; we will write $\star : A$ for the unique element of any $\top$-sealed type $A$.

Intuitively, a $\phi$-transparent type is one that "thinks" $\phi$ is true; conversely, a $\phi$-sealed type is one that "thinks" $\phi$ is false and thus contracts to a point under $\phi$. The $\phi$-transparent and $\phi$-sealed types are each governed by modalities, referred to as ***open*** and ***closed*** respectively.

*Definition 4.4 (Open modality).* We will write $\phi \Rightarrow A$ for the *implicit* function space $\phi \to A$, which we refer to as the ***open modality*** for $\phi$; we will leave both the $\lambda$-abstraction and application implicit in our notation. Likewise, given an element $A : \phi \Rightarrow \mathbb{U}$ we will write $\phi \Rightarrow A$ for the *implicit* dependent product of $A$; this is the ***dependent open modality***.[5]

*Definition 4.5 (Closed modality).* We will write $\phi \bullet A$ for the following quotient inductive type, which we shall call the ***closed modality*** associated to $\phi$:

$$\begin{array}{l} \textbf{quotient data } \phi \bullet A : \mathbb{U} \textbf{ where} \\ \eta_{\phi\bullet} : A \to \phi \bullet A \\ \star : \{\_ : \phi\} \to \phi \bullet A \\ \_ : \{\_ : \phi, u : \phi \bullet A\} \to u = \star \end{array}$$

(We use curly braces to indicate *implicit* arguments.) Put another way, $\phi \bullet A$ is the quotient of the coproduct $\phi + A$ under the $\mathbb{P}$-valued equivalence relation $u \sim v \iff \phi \lor (u = v)$. The elimination

[5] The implicitness is only a matter of convenient notation; it plays no mathematical role.

rule for $\phi \bullet A$ is thus a case statement with a side condition:

$$\frac{x : \phi \bullet A \vdash Cx : \mathbb{U} \qquad x : A \vdash fx : C\,(\eta x) \qquad \_ : \phi \vdash g : C\star \qquad x : A, \_ : \phi \vdash fx = g : C\star \qquad u : \phi \bullet A}{\mathbf{case}\ u\ \mathbf{of}\ \eta_{\phi\bullet}x \hookrightarrow fx \mid \star \hookrightarrow g : Cu}$$

Note that we have only assumed an elimination rule for motives valued in $\mathbb{U}$.

*Remark 4.6 (Effectivity).* Any equivalence relation valued in $\mathbb{P}$ is effective, as $\mathbb{P}$ satisfies propositional extensionality; thus the quotient in Definition 4.5 is effective. The only subtlety worth noting is that although the equivalence relation $\phi \vee (u = v)$ is by definition a pushout in $\mathbb{P}$ of the projections $\phi \leftarrow \phi \wedge (u = v) \rightarrow u = v$, this pushout *need not* be preserved by the inclusion $\mathbb{P} \hookrightarrow \mathbb{U}$; an example of this behavior in semantics is that the union of two regular subobjects need not be regular.

**Fact 4.7.** *A type $A : \mathbb{U}$ is $\phi$-transparent if and only if it is modal for the open modality $\phi \Rightarrow -$; the type $A : \mathbb{U}$ is $\phi$-sealed if and only if it is modal for the closed modality $\phi \bullet -$.*

In light of Fact 4.7, it is instructive to point out that the elimination form for the closed modality $(x \leftarrow u; fx)$ can be implemented by $\mathbf{case}\ u\ \mathbf{of}\ \eta_{\phi\bullet}x \hookrightarrow fx \mid \star \hookrightarrow \star$.

*Remark 4.8.* In other applications of LRAT such as those of Sterling and Angiuli [2021]; Sterling and Harper [2022], case splits were included for $\phi \vee \psi$ even when $\phi$ and $\psi$ are not disjoint. This is possible in the internal language of a topos, but *not* in the **iGDTT** where the universe of propositions is meant to lie within $\mathbb{V}_i$, which will be interpreted by a quasitopos rather than a true topos. Roughly the issue is that in a quasitopos $\mathscr{E}$, the union of two regular subobjects need not be regular unless $\mathscr{E}$ is quasiadhesive [Johnstone et al. 2007].

*Extension types.* Given a type $A$, a proposition $\phi$ and an element $a : \phi \Rightarrow A$ we will write $\{A \mid \phi \hookrightarrow a\}$ for the subtype $\{x : A \mid \phi \Rightarrow x = a\} \subseteq A$. We will write $\{A \mid \phi \hookrightarrow a, \psi \hookrightarrow b, \ldots\}$ for the extension type $\{A \mid \phi \vee \psi \vee \ldots \hookrightarrow [\phi \hookrightarrow a, \psi \hookrightarrow b, \ldots]\}$ when $\phi, \psi$ are disjoint.

*Remark 4.9 (LRAT axiomatizes the parametricity translation).* In the synthetic setting, to each type $A$ we may associate the span $(\Phi_{\mathrm{L}} \Rightarrow A) \longleftarrow A \longrightarrow (\Phi_{\mathrm{R}} \Rightarrow A)$ given by the units of the open modalities for $\Phi_{\mathrm{L}}, \Phi_{\mathrm{R}}$; this should be thought of as the span that $A$ is taken to in the binary parametricity translation. Indeed, given $u_{\mathrm{L}} : \Phi_{\mathrm{L}} \Rightarrow A$ and $u_{\mathrm{R}} : \Phi_{\mathrm{R}} \Rightarrow A$ the subtype $\{A \mid \Phi_{\mathrm{L}} \hookrightarrow u_{\mathrm{L}}, \Phi_{\mathrm{R}} \hookrightarrow u_{\mathrm{R}}\}$ can be thought of as the type of proofs that $u_{\mathrm{L}}$ and $u_{\mathrm{R}}$ are related in the correspondence. It is in this sense that LRAT axiomatizes the parametricity translation.

A final axiom of the synthetic relational extension of **iGDTT** is the *refinement type*.[6] Given a type $A : \Phi \Rightarrow \mathbb{U}$ and a family of $\Phi$-sealed types $B : (\Phi \Rightarrow A) \rightarrow \mathbb{U}$, we may consider $\sum_{x:\Phi\Rightarrow A} Bx$. The ***refinement type*** is a new code for this dependent sum, written $[\Phi \hookrightarrow x : A \mid Bx]$, that enjoys the following *additional* principle of type equality: the refinement type becomes equal to $A$ in the presence of $\_ : \Phi$. In other words, the refinement type is governed by the interface presented in Fig. 1. We have not assumed that any proposition $\phi : \mathbb{P}$ besides $\Phi$ is equipped with a refinement type connective.

*Notation 4.10 (Copattern notation).* We will often define elements of $[\Phi \hookrightarrow x : A \mid Bx]$ using Agda-style "copattern" notation; for instance, the pair of declarations on the left is meant to denote

[6] In prior presentations of synthetic logical relations [Sterling and Angiuli 2021; Sterling and Harper 2021], the role played here by the refinement type connective (first introduced by Sterling and Harper [2022]) was instead played by the so-called *realignment* axiom. The two are interderivable [Gratzer et al. 2022, §5], but the refinement type connective is both more direct and provides stronger geometrical intuition.

$$\textsc{formation}\quad \frac{A : \Phi \Rightarrow A \qquad B : (\Phi \Rightarrow A) \to \mathbb{U} \qquad \Phi \Rightarrow \forall x : A.\, Bx \cong 1}{[\Phi \hookrightarrow x : A \mid Bx] : \{\mathbb{U} \mid \Phi \hookrightarrow A\}}$$

$$\textsc{introduction}\quad \frac{\ldots \qquad a : \Phi \Rightarrow A \qquad b : Ba}{[\Phi \hookrightarrow a \mid b] : \{[\Phi \hookrightarrow x : A \mid Bx] \mid \Phi \hookrightarrow a\}} \qquad \textsc{elimination}\quad \frac{\ldots \qquad u : [\Phi \hookrightarrow x : A \mid Bx]}{\underline{u} : Bu}$$

$$\textsc{computation}\quad \frac{\ldots \qquad a : \Phi \Rightarrow A \qquad b : Ba}{\underline{[\Phi \hookrightarrow a \mid b]} = b : Ba} \qquad \textsc{uniqueness}\quad \frac{\ldots \qquad u : [\Phi \hookrightarrow x : A \mid Bx]}{u = [\Phi \hookrightarrow u \mid \underline{u}] : [\Phi \hookrightarrow x : A \mid Bx]}$$

Fig. 1. The interface to the *refinement type* connective.

the single declaration on the right:

$$\begin{bmatrix} \Phi \hookrightarrow u \triangleq a \\ \underline{u} \triangleq b \end{bmatrix} \rightsquigarrow u \triangleq [\Phi \hookrightarrow a \mid b]$$

### 4.2 The state monad in $\mathbf{iGDTT}^{\mathsf{ref}}_{\mathrm{LRAT}}$

In Section 3.1, we extended **iGDTT** with general reference types and a state monad $\mathsf{T}$. In this section, we add to the axiomatization of $\mathbf{iGDTT}^{\mathsf{ref}}_{\mathrm{LRAT}}$ constructs governing the weak bisimulation of stateful computations. The following two rules enable exhibiting a correspondence between two computations that take different numbers of steps:

$$\frac{}{\mathsf{step}_{\mathrm{L}} : \{\mathsf{T}\,1 \mid \Phi_{\mathrm{L}} \hookrightarrow \mathsf{step}, \Phi_{\mathrm{R}} \hookrightarrow \mathsf{ret}\,()\}} \qquad \frac{}{\mathsf{step}_{\mathrm{R}} : \{\mathsf{T}\,1 \mid \Phi_{\mathrm{L}} \hookrightarrow \mathsf{ret}\,(), \Phi_{\mathrm{R}} \hookrightarrow \mathsf{step}\}}$$

*Example 4.11.* With the above in hand, it is possible exhibit a correspondence between two computations that access the memory different numbers of times. Let $r : \mathsf{ref}\,\mathbb{Z}$ and consider the programs $M_{\mathrm{L}}, M_{\mathrm{R}}$ defined below:

$$\begin{aligned} M_{\mathrm{L}} &\triangleq x \leftarrow \mathsf{get}_{\mathbb{Z}} r; \mathsf{set}_{\mathbb{Z}}\, r\, (x+1); y \leftarrow \mathsf{get}_{\mathbb{Z}} r; \mathsf{set}_{\mathbb{Z}}\, r\, (y+1); \mathsf{ret}\,() \\ M_{\mathrm{R}} &\triangleq x \leftarrow \mathsf{get}_{\mathbb{Z}} r; \mathsf{set}_{\mathbb{Z}}\, r\, (x+2); \mathsf{ret}\,() \end{aligned}$$

We may define a correspondence $\mathcal{M} : \{\mathsf{T}\,1 \mid \Phi_{\mathrm{L}} \hookrightarrow M_{\mathrm{L}}, \Phi_{\mathrm{R}} \hookrightarrow M_{\mathrm{R}}\}$ as follows:

$$\mathcal{M} \triangleq x \leftarrow \mathsf{get}_{\mathbb{Z}} r; \mathsf{step}_{\mathrm{L}}; \mathsf{set}_{\mathbb{Z}}\, r\, (x+2); \mathsf{ret}\,()$$

While $\mathcal{M}$ is evidently of type $\mathsf{T}\,1$, it remains to argue that it is equal to $M_{\mathrm{L}}$ (resp. $M_{\mathrm{R}}$) under the assumption $\Phi_{\mathrm{L}}$ (resp. $\Phi_{\mathrm{R}}$). Both claims follow from equational reasoning; assuming $\Phi_{\mathrm{L}}$ we compute:

$$\begin{aligned} \Phi_{\mathrm{L}} \hookrightarrow M_{\mathrm{L}} &= x \leftarrow \mathsf{get}_{\mathbb{Z}} r; \mathsf{set}_{\mathbb{Z}}\, r\, (x+1); y \leftarrow \mathsf{get}_{\mathbb{Z}} r; \mathsf{set}_{\mathbb{Z}}\, r\, (y+1); \mathsf{ret}\,() \\ &= x \leftarrow \mathsf{get}_{\mathbb{Z}} r; \mathsf{step}; \mathsf{set}_{\mathbb{Z}}\, r\, (x+1); \mathsf{set}_{\mathbb{Z}}\, r\, (x+1+1); \mathsf{ret}\,() \\ &= x \leftarrow \mathsf{get}_{\mathbb{Z}} r; \mathsf{step}; \mathsf{set}_{\mathbb{Z}}\, r\, (x+1+1); \mathsf{ret}\,() \\ &= x \leftarrow \mathsf{get}_{\mathbb{Z}} r; \mathsf{step}; \mathsf{set}_{\mathbb{Z}}\, r\, (x+2); \mathsf{ret}\,() \\ &= x \leftarrow \mathsf{get}_{\mathbb{Z}} r; \mathsf{step}_{\mathrm{L}}; \mathsf{set}_{\mathbb{Z}}\, r\, (x+2); \mathsf{ret}\,() \\ &= \mathcal{M} \end{aligned}$$

Likewise, assuming $\Phi_{\mathrm{R}}$ we have the following:

$$\Phi_{\mathrm{R}} \hookrightarrow M_{\mathrm{R}} = x \leftarrow \mathsf{get}_{\mathbb{Z}} r; \mathsf{set}_{\mathbb{Z}}\, r\, (x+2); \mathsf{ret}\,() = x \leftarrow \mathsf{get}_{\mathbb{Z}} r; \mathsf{step}_{\mathrm{L}}; \mathsf{set}_{\mathbb{Z}}\, r\, (x+2); \mathsf{ret}\,() = \mathcal{M}$$

### 4.3 Case study: local references and closures

In this section we consider a synthetic version of an example from Birkedal et al. [2010, §6.3] concerning a correspondence between two imperative counter modules constructed using a local reference and two closures. Whereas *op. cit.* needed to work inside the model, the combination of dependent types and synthetic relational constructs enables us to work directly in the type theory. Consider the following two implementations of an imperative counter module using local references, one of which counts up and the other counts down:

$$\begin{array}{ll}
M_{\mathrm{L}}, M_{\mathrm{R}} : \mathsf{T}\,(\mathsf{T}\,1 \times \mathsf{T}\,\mathbb{Z}) & \mathsf{incr}, \mathsf{decr} : \mathsf{ref}\,\mathbb{Z} \to \mathsf{T}\,1 \\
M_{\mathrm{L}} \triangleq (r \leftarrow \mathsf{new}_{\mathbb{Z}}\,0; \mathsf{ret}\,(\mathsf{incr}\,r, \mathsf{get}_{\mathbb{Z}} r)) & \mathsf{incr}\,r \triangleq x \leftarrow \mathsf{get}_{\mathbb{Z}} r; \mathsf{set}_{\mathbb{Z}}\,r\,(x+1) \\
M_{\mathrm{R}} \triangleq (r \leftarrow \mathsf{new}_{\mathbb{Z}}\,0; \mathsf{ret}\,(\mathsf{decr}\,r, x \leftarrow \mathsf{get}_{\mathbb{Z}} r; \mathsf{ret}\,(-x))) & \mathsf{decr}\,r \triangleq x \leftarrow \mathsf{get}_{\mathbb{Z}} r; \mathsf{set}_{\mathbb{Z}}\,r\,(x-1)
\end{array}$$

We wish to construct a correspondence between $M_{\mathrm{L}}$ and $M_{\mathrm{R}}$, *i.e.* an element $\mathcal{M} : \{\mathsf{T}\,(\mathsf{T}\,1 \times \mathsf{T}\,\mathbb{Z}) \mid \Phi_{\mathrm{L}} \hookrightarrow M_{\mathrm{L}}, \Phi_{\mathrm{R}} \hookrightarrow M_{\mathrm{R}}\}$. To do so, we define a correspondence on $\mathbb{Z}$ describing the heap invariant using a refinement type:

$$\begin{array}{l}
\mathcal{Z} : \{\mathbb{U} \mid \Phi \hookrightarrow \mathbb{Z}\} \\
\mathcal{Z} \triangleq [\Phi \hookrightarrow x : \mathbb{Z} \mid \Phi \bullet \{(i_{\mathrm{L}} : \{\mathbb{Z} \mid \Phi_{\mathrm{L}} \hookrightarrow x\}, i_{\mathrm{R}} : \{\mathbb{Z} \mid \Phi_{\mathrm{R}} \hookrightarrow x\}) \mid i_{\mathrm{L}} = -i_{\mathrm{R}}\}]
\end{array}$$

*Exegesis 4.12.* It is important to understand why the invariant $\mathcal{Z}$ is defined the way it is.

(1) First we attach an element $x : \Phi \Rightarrow \mathbb{Z}$, which is in particular a pair of a "left" element $x_{\mathrm{L}} : \Phi_{\mathrm{L}} \Rightarrow \mathbb{Z}$ and a "right" element $x_{\mathrm{R}} : \Phi_{\mathrm{R}} \Rightarrow \mathbb{Z}$ with no conditions whatsoever.

(2) We wish to assert that $x_{\mathrm{R}}$ is the negation of $x_{\mathrm{L}}$; but this is not type correct, because these partial integers lie in disjoint worlds.

(3) Instead we will glue onto the partial integer $x = [\Phi_{\mathrm{L}} \hookrightarrow x_{\mathrm{L}}, \Phi_{\mathrm{R}} \hookrightarrow x_{\mathrm{R}}]$ a pair of total integers $i_{\mathrm{L}}, i_{\mathrm{R}} : \mathbb{Z}$ restricting under $\Phi_{\mathrm{L}}, \Phi_{\mathrm{R}}$ to $x_{\mathrm{L}}, x_{\mathrm{R}}$ such that $i_{\mathrm{L}} = -i_{\mathrm{R}}$.

(4) It is necessary to hide the attachment of $(i_{\mathrm{L}}, i_{\mathrm{R}})$ to $x$ underneath the closed modality $\Phi \bullet -$, since otherwise $\mathcal{Z}$ would restrict to strictly more than just $\mathbb{Z}$ under $\Phi$. Without the closed modality, the formation rule for the refinement type employed here would not apply.

With the invariant $\mathcal{Z}$ in hand, we will construct a *new* counter implementation $\mathcal{M}$ that allocates an element of $\mathcal{Z}$; this new counter implementation should restrict "on the left" to $M_{\mathrm{L}}$ and "on the right" to $M_{\mathrm{R}}$. We divide the construction into three main subroutines $\mathsf{init}_{\mathcal{M}}$, $\mathsf{tick}_{\mathcal{M}}$, and $\mathsf{read}_{\mathcal{M}}$:

$$\begin{array}{l}
\mathcal{M} : \{\mathsf{T}\,(\mathsf{T}\,1 \times \mathsf{T}\,\mathbb{Z}) \mid \Phi_{\mathrm{L}} \hookrightarrow M_{\mathrm{L}}, \Phi_{\mathrm{R}} \hookrightarrow M_{\mathrm{R}},\} \\
\mathcal{M} \triangleq (r \leftarrow \mathsf{new}_{\mathcal{Z}}\,\mathsf{init}_{\mathcal{M}}; \mathsf{ret}\,(\mathsf{tick}_{\mathcal{M}}\,r, \mathsf{read}_{\mathcal{M}}\,r))
\end{array}$$

For the initialization, it is simple enough to construct an element of $\mathcal{Z}$ showing that $0 = -0$.

$$\begin{array}{l}
\mathsf{init}_{\mathcal{M}} : \{\mathcal{Z} \mid \Phi \hookrightarrow 0\} \\
\Phi \hookrightarrow \mathsf{init}_{\mathcal{M}} \triangleq 0 \\
\underline{\mathsf{init}_{\mathcal{M}}} \triangleq \eta_{\Phi\bullet}(0, 0)
\end{array}$$

Next we describe the operation that ticks the counter, factoring through an auxiliary operation $\mathsf{upd}_{\mathcal{M}}$ that witnesses the preservation of the $\mathcal{Z}$ correspondence by the change to the reference cell.

$$\begin{array}{ll}
\mathsf{tick}_{\mathcal{M}} : \{\mathsf{ref}\,\mathcal{Z} \to \mathsf{T}\,1 \mid \Phi_{\mathrm{L}} \hookrightarrow \mathsf{incr}, \Phi_{\mathrm{R}} \hookrightarrow \mathsf{decr}\} & \mathsf{upd}_{\mathcal{M}} : \{\mathcal{Z} \to \mathcal{Z} \mid \Phi_{\mathrm{L}} \hookrightarrow \lambda x.x+1, \Phi_{\mathrm{R}} \hookrightarrow \lambda x.x-1\} \\
\mathsf{tick}_{\mathcal{M}} r \triangleq x \leftarrow \mathsf{get}_{\mathcal{Z}} r; \mathsf{set}_{\mathcal{Z}}\,r\,(\mathsf{upd}_{\mathcal{M}} x) & \Phi_{\mathrm{L}} \hookrightarrow \mathsf{upd}_{\mathcal{M}} x \triangleq x + 1 \\
 & \Phi_{\mathrm{R}} \hookrightarrow \mathsf{upd}_{\mathcal{M}} x \triangleq x - 1 \\
 & \underline{\mathsf{upd}_{\mathcal{M}} x} \triangleq (i_{\mathrm{L}}, i_{\mathrm{R}}) \leftarrow \underline{x}; \eta_{\Phi\bullet}(i_{\mathrm{L}} + 1, i_{\mathrm{R}} - 1)
\end{array}$$

In the definition of $\underline{\mathsf{upd}_{\mathcal{M}} x}$, we have used the Kleisli extension for the closed modality $\Phi \bullet -$. The well-typedness of our definition follows from the fact that $i_{\mathrm{L}} = -i_{\mathrm{R}}$ implies $i_{\mathrm{L}} + 1 = -(i_{\mathrm{R}} - 1)$.

Finally we construct a function to read the contents of the local state, factoring through a function $\mathsf{tidy}_{\mathcal{M}}$ that verifies the correspondence between $x$ and $-x$ in the output:

$$\begin{array}{l}
\mathsf{read}_{\mathcal{M}} : \{\mathsf{ref}\,\mathcal{Z} \to \mathsf{T}\,\mathbb{Z} \mid \mathbf{\Phi}_{\mathrm{L}} \hookrightarrow \lambda r.\mathsf{get}_{\mathbb{Z}} r, \mathbf{\Phi}_{\mathrm{R}} \hookrightarrow \lambda r.x \leftarrow \mathsf{get}_{\mathbb{Z}} r; \mathsf{ret}\,(-x)\} \\
\mathsf{read}_{\mathcal{M}} r \triangleq x \leftarrow \mathsf{get}_{\mathcal{Z}} r; \mathsf{ret}\,(\mathsf{tidy}_{\mathcal{M}} x) \\
\\
\mathsf{tidy}_{\mathcal{M}} : \{\mathcal{Z} \to \mathbb{Z} \mid \mathbf{\Phi}_{\mathrm{L}} \hookrightarrow \lambda x.x, \mathbf{\Phi}_{\mathrm{R}} \hookrightarrow \lambda x.-x\} \\
\mathsf{tidy}_{\mathcal{M}} x \triangleq \\
\quad \mathbf{case}\ \underline{x}\ \mathbf{of} \\
\qquad \eta_{\mathbf{\Phi}\bullet}(i_{\mathrm{L}}, i_{\mathrm{R}}) \hookrightarrow i_{\mathrm{L}} \\
\qquad \star \hookrightarrow [\mathbf{\Phi}_{\mathrm{L}} \hookrightarrow x, \mathbf{\Phi}_{\mathrm{R}} \hookrightarrow -x]
\end{array}$$

*Bisimulation of denotations.* By interpreting the correspondence $\mathcal{M}$ in our semantic model (see Section 5), it will follow that the results of initializing $M_{\mathrm{L}}, M_{\mathrm{R}}$ with any heap are *weakly bisimilar*, *i.e.* equal up to computation steps. In fact, this particular example exhibits a *strong* bisimulation.

### 4.4 Case study: correspondences in abstract data types

We may adapt our previous example to exhibit a simulation between two implementations of an abstract data type for imperative counters as in Birkedal et al. [2010, §6.2]. In particular, consider the following abstract data type:

$$\mathsf{COUNTER} \triangleq \exists_{\alpha:\mathbb{U}}\, \mathsf{T}\,\alpha \times (\alpha \to \mathsf{T}\,1) \times (\alpha \to \mathsf{T}\,\mathbb{Z})$$

We consider the following two implementations of COUNTER:

$$\begin{array}{l}
M_{\mathrm{L}} \triangleq \mathsf{pack}\,(\mathsf{ref}\,\mathbb{Z}, (\mathsf{new}_{\mathbb{Z}}\,0, \mathsf{incr}, \lambda r.\mathsf{get}_{\mathbb{Z}} r)) \\
M_{\mathrm{R}} \triangleq \mathsf{pack}\,(\mathsf{ref}\,\mathbb{Z}, (\mathsf{new}_{\mathbb{Z}}\,0, \mathsf{decr}, \lambda r.x \leftarrow \mathsf{get}_{\mathbb{Z}} r; \mathsf{ret}\,(-x)))
\end{array}$$

We may construct a correspondence between these two imperative counter structures as follows, re-using the constructions of Section 4.3.

$$\begin{array}{l}
\mathcal{M} : \{\mathsf{COUNTER} \mid \mathbf{\Phi}_{\mathrm{L}} \hookrightarrow M_{\mathrm{L}}, \mathbf{\Phi}_{\mathrm{R}} \hookrightarrow M_{\mathrm{R}}\} \\
\mathcal{M} = \mathsf{pack}\,(\mathsf{ref}\,\mathcal{Z}, (\mathsf{new}_{\mathcal{Z}}\,\mathsf{init}_{\mathcal{M}}, \mathsf{tick}_{\mathcal{M}}, \mathsf{read}_{\mathcal{M}}))
\end{array}$$

*Bisimulation of denotations.* As in Section 4.3, by interpreting the correspondence $\mathcal{M}$ into the semantic model of Section 5 we may exhibit a weak (in fact, strong) bisimulation between the denotations of $M_{\mathrm{L}}$ and $M_{\mathrm{R}}$ when initialized with any heap.

## 5 SEMANTIC MODELS OF iGDTT AND $\mathbf{iGDTT}^{\mathsf{ref}}_{\mathrm{LRAT}}$

Thus far we have introduced a series of type theories—**iGDTT**, $\mathbf{iGDTT}^{\mathsf{ref}}$, and $\mathbf{iGDTT}^{\mathsf{ref}}_{\mathrm{LRAT}}$—but we have not yet proven that these type theories are consistent. In this section we justify all three theories by constructing semantic models.

In Section 5.1 we show to build a model of **iGDTT** based combining realizability and the topos of trees. In Section 5.2, we show that presheaves internal to a model of **iGDTT** also assemble into a model of **iGDTT**. Using this, we then lift a base model to a model of $\mathbf{iGDTT}^{\mathsf{ref}}_{\mathrm{LRAT}}$ in Section 5.4 by taking presheaves on a carefully chosen category and constructing a more sophisticated version of the possible-worlds model introduced in Section 2.4. It proves most convenient to pass directly from a model of **iGDTT** to a model of $\mathbf{iGDTT}^{\mathsf{ref}}_{\mathrm{LRAT}}$ rather than constructing an intermediate model of $\mathbf{iGDTT}^{\mathsf{ref}}$. Of course, any model of $\mathbf{iGDTT}^{\mathsf{ref}}_{\mathrm{LRAT}}$ is (by restriction) a model of $\mathbf{iGDTT}^{\mathsf{ref}}$. We return, however, to the construction of a model specifically of $\mathbf{iGDTT}^{\mathsf{ref}}$ directly from **iGDTT** in a far more general context in Section 6.

The construction of this sequence of models immediately yields the consistency of all three type theories. The particular models, however, show more than this. We use our model constructions to argue that a relation constructed in $\mathbf{iGDTT}^{\mathsf{ref}}_{\mathrm{LRAT}}$ induces a weak bisimulation between the related programs in the induced model of $\mathbf{iGDTT}^{\mathsf{ref}}$.

*Remark 5.1.* Unlike the rest of the paper, throughout this section we assume knowledge of category theory. In particular, in addition to the basic concepts of category theory we assume some familiarity with categorical realizability (assemblies, modest sets, *etc.*). We refer the reader to Streicher [2017]; van Oosten [2008] for a thorough introduction.

## 5.1 Constructing base models of iGDTT

A concrete model of **iGDTT** can be constructed using a combination of realizability and presheaves. Rather than belabor the already well-studied interpretation of dependent type theory into well-adapted categories (*e.g.* locally cartesian closed categories, topoi, *etc.*) [Hofmann 1997], we describe and exhibit the relevant categorical structure needed to interpret the nonstandard aspects of **iGDTT**: impredicative universes and guarded recursion.

Let $\mathcal{S}$ be a realizability topos and let $(\mathbb{O}, \leq, \prec)$ be a ***separated intuitionistic well-founded poset*** in $\mathcal{S}$, *i.e.* a poset $(\mathbb{O}, \leq)$ equipped with a transitive subrelation $\prec \subseteq \leq \subseteq \mathbb{O} \times \mathbb{O}$ satisfying the following additional conditions:

(1) *separation*: the object $\mathbb{O}$ is an assembly and both $\leq, \prec$ are regular subobjects,
(2) *left compatibility*: if $u \leq v$ and $v \prec w$ then $u \prec w$,
(3) *right compatibility*: if $u \prec v$ and $v \leq w$ then $u \prec w$,
(4) *well-foundedness*: every element of $\mathbb{O}$ is $\prec$-accessible, where the $\prec$-accessible elements are defined to be the smallest subset $I \subseteq \mathbb{O}$ such that if $v \in I$ for all $v \prec u$, then $u \in I$.

*Remark 5.2.* In classical mathematics, the appropriate notion of well-founded poset is considerably simpler as we tend to define $x \prec y \Leftrightarrow x \leq y \wedge x \neq y$. This simpler style of well-founded order is particularly inappropriate for intuitionistic mathematics (*e.g.* the mathematics of a realizability topos), in which $\mathbb{O}$ need not have decidable equality. Our definition of intuitionistic well-founded posets is inspired by Taylor's analysis of intuitionistic ordinals [Taylor 1996].

Given a hierarchy of Grothendieck universes $\mathcal{U}_i$ in **Set** such that $\Gamma\mathbb{O} \in \mathcal{U}_0$, we conclude:

THEOREM 5.3. *The category of internal diagrams*[7] $\mathscr{E} = \left[\mathbb{O}^{\circ}, \mathcal{S}\right]$ *is a model of* **iGDTT** *in which:*

(1) *the predicative universes* $\mathbb{V}_i$ *are modelled by the Hofmann–Streicher liftings [Hofmann and Streicher 1997] of the universes of* $\mathcal{U}_i$*-small assemblies in* $\mathcal{S}$*,*
(2) *the impredicative universes* $\mathbb{P}, \mathbb{U} \in \mathbb{V}_i$ *is modelled by the Hofmann–Streicher liftings of the universes of* $\neg\neg$*-closed propositions and of modest sets in* $\mathcal{S}$ *respectively,*
(3) *the later modality* $\blacktriangleright$ *is computed explicitly by the equation* $(\blacktriangleright A)u = \varprojlim_{v \prec u} Av$.

PROOF. The guarded recursive aspects follow from the general results of Palombi and Sterling [2022]; for the rest, it can be seen that the Hofmann–Streicher lifting of a pair of universes preserves impredicativity of the lower universe when $\mathbb{O}$ is internal to the upper universe. Our interpretation of $\mathbb{P}$ is automatically univalent and satisfies proof irrelevance, and the only subtle point is that it is closed under equality. This follows because every type classified by either $\mathbb{U}$ or $\mathbb{V}$ is $\neg\neg$-separated (since they are assemblies), which means precisely that their equality predicates are $\neg\neg$-closed. □

We describe the "standard" instantiation of Theorem 5.3 in Example 5.4 below.

[7] We refer to Borceux [1994, §8.1] for discussion of internal diagrams.

*Example 5.4 (Standard model of* **iGDTT***).* Let **Eff** be the *effective topos* [Hyland 1982], namely the realizability topos constructed from Kleene's first algebra; there is a modest intuitionistic well-founded poset $\omega$ in **Eff** given by the natural numbers object of the topos under inequality and strict inequality. Then $\left[\omega^\circ, \mathbf{Eff}\right]$ is model of **iGDTT** according to Theorem 5.3.

From the non-emptiness of $\omega$ it follows immediately that $\left[\omega^\circ, \mathbf{Eff}\right]$ is a non-trivial topos, *i.e.* one in which the initial object is not inhabited. Thus we obtain the following corollary:

Corollary 5.5. **iGDTT** *is consistent.*

*Remark 5.6 (Strong and weak completeness).* It is important that each $\mathbb{V}_i$ be built from a subuniverse of small *assemblies* rather than of more general types in $\mathcal{S}$: the full internal subcategory spanned by modest sets is strongly complete over assemblies, but only *weakly* complete over the rest of $\mathcal{S}$ [Hyland et al. 1990]. Weak completeness is insufficient to interpret $\forall$.

### 5.2 The Hofmann–Streicher lifting of a model of iGDTT

Let $\mathscr{E}$ be an elementary topos model of **iGDTT** *e.g.*, Example 5.4; we will show that for certain internal categories $\mathbb{C}$ in $\mathscr{E}$, the topos of internal diagrams $[\mathbb{C}, \mathscr{E}]$ is a model of **iGDTT**. To avoid confusion, we will write things like $\mathbb{U}, \mathbb{V}_i$ for constructs of $\mathscr{E}$ and $\overline{\mathbb{U}}, \overline{\mathbb{V}_i}$ for the corresponding constructs that we will define in the category of internal diagrams $[\mathbb{C}, \mathscr{E}]$.

Let $\mathbb{C}$ be a category internal to $\mathbb{V}_0$, *i.e.* a category object in $\mathscr{E}$ whose object of objects and hom objects are classified by the universe $\mathbb{V}_0$. Then following Hofmann and Streicher [1997] we may lift each universe $\mathbb{X} \in \{\mathbb{P}, \mathbb{U}, \mathbb{V}_i\}$ of $\mathscr{E}$ to a universe $\overline{\mathbb{X}}$ in $[\mathbb{C}, \mathscr{E}]$, setting $\overline{\mathbb{X}}c = [c \downarrow \mathbb{C}, \mathbb{X}]$. The only surprise is that $\overline{\mathbb{U}}$ remains classified by $\overline{\mathbb{V}_i}$, considering the fact that the coslice $c \downarrow \mathbb{C}$ is large relative to $\mathbb{U}$. Nevertheless we have $\overline{\mathbb{U}} \in \overline{\mathbb{V}_i}$ because $\mathbb{U}$ is impredicative in $\mathbb{V}_i$; by an explicit computation, it follows that $\overline{\mathbb{U}}$ can be encoded by an element of $\overline{\mathbb{V}_i}$ using only $\forall$, $\sum$, and $(=)$ in each fiber. The impredicativity of $\mathbb{U}$ in $\mathbb{V}_i$ likewise ensures that $\overline{\mathbb{U}}$ is impredicative over $\overline{\mathbb{V}_i}$.

Closure under the constructs of guarded dependent type theory then follows once again from the general results of Palombi and Sterling [2022] concerning the pointwise lifting of guarded recursion from the base into presheaves. It can also be seen that good properties of the later modality are preserved by this lifting; if $\mathscr{E}$ is ***globally adequate*** in the sense of *op. cit.* and $\mathbb{C}$ has an initial object, then the $[\mathbb{C}, \mathscr{E}]$ is also globally adequate. Global adequacy means that the global points of $\blacktriangleright\mathbb{N}$ are the actual natural numbers. Summarizing:

Lemma 5.7. *Let $\mathscr{E}$ be an elementary topos model of* **iGDTT** *and let $\mathbb{C}$ be a category internal to $\mathbb{V}_0 \in \mathscr{E}$. The category of internal diagrams $[\mathbb{C}, \mathscr{E}]$ is a model of* **iGDTT** *in which:*

1. *all universes are modelled by the Hofmann–Streicher liftings [Hofmann and Streicher 1997] of the corresponding universes in $\mathscr{E}$;*
2. *the later modality $\blacktriangleright$ is computed pointwise,* i.e. *we have $(\blacktriangleright A)c = \blacktriangleright(Ac)$.*

*Strict extension structures.* In order to construct the store model in Section 5.4, we will require that the impredicative universe $\mathbb{U}$ also supports a *strict extension structure.* While we require this property in only one concrete setting, it is most natural to define with respect to an arbitrary *dominance* $1_{\mathscr{E}} \rightarrowtail \Sigma \in \mathscr{E}$, *i.e.* a universe of propositions closed under truth and dependent conjunction.

Recall that for any $\phi : \Sigma$, a partial element $A : \phi \Rightarrow \mathbb{U}$ can be extended to a total element $B : \mathbb{U}$ that restricts to $A$ up to isomorphism *e.g.*, by setting $B$ to $\phi \Rightarrow A$. In Section 5.4, however, we require the ability to choose an extension which agrees exactly with $A$ under $\phi$. We say a universe $\mathbb{X}$ has a ***strict extension structure*** when this is possible *i.e.*, when it validates following rules:

$$\textsc{strict extension}\quad \frac{\phi : \Sigma \qquad A : \phi \Rightarrow \mathbb{X}}{\phi_* A : \mathbb{X}} \qquad\qquad \textsc{strict extension intro}\quad \frac{\phi : \Sigma \qquad A : \phi \Rightarrow \mathbb{X} \qquad a : \phi \Rightarrow A}{\langle \phi \rangle a : \phi_* A}$$

$$\frac{\phi : \Sigma \qquad A : \phi \Rightarrow \mathbb{X} \qquad \_ : \phi}{\phi_* A \equiv A : \mathbb{X}} \qquad\qquad \frac{\phi : \Sigma \qquad A : \phi \Rightarrow \mathbb{X} \qquad a : \phi \Rightarrow A \qquad \_ : \phi}{\langle \phi \rangle a \equiv a : A}$$

Fortunately, given a dominance $\Sigma$ we are always able to replace a universe $\mathbb{X}$ with a new universe equipped with an extension structure without perturbing the collection of types it classifies (Lemma 5.9). The idea is to replace $\mathbb{X}$ with its $\Sigma$-partial map classifier.

*Definition 5.8 (Partial element classifier).* Let $A$ be a type and let $\Sigma$ be a dominance; the ***$\Sigma$-partial map classifier*** $A^+$ is defined to be the dependent sum $\sum_{\phi:\Sigma} \phi \Rightarrow A$. We have a natural transformation $\eta : \mathrm{id} \longrightarrow (-)^+$ sending each $a : A$ to the pair $(\top, a) : A^+$.

For any universe $\mathbb{X}$ that classifies each $\phi : \Sigma$, we have an algebra structure $\Pi : \mathbb{X}^+ \longrightarrow \mathbb{X}$ for the raw endofunctor $-^+$, sending each $A : \mathbb{X}^+$ to its dependent product $\Pi Q = \pi_1 Q \Rightarrow \pi_2 Q$. Viewing $\mathbb{X}$ as an internal groupoid, we see that $\Pi$ is a "pseudo-algebra" structure for the **pointed** endofunctor $-^+$, as each $\Pi(\top, A)$ is isomorphic to $A$.[8] It can be seen that a strict extension structure for $\mathbb{X}$ is exactly the same thing as a strictification of $\Pi$, *i.e.* a strict algebra for the pointed endofunctor $-^+$ that is additionally isomorphic to $\Pi$.[9]

Lemma 5.9 (Strictification). *Let $\Sigma$ be a dominance such that each $\phi : \Sigma$ is classified by $\mathbb{X}$; then there exists a universe $\mathbb{X}^s$ equipped with a strict extension structure for $\Sigma$ such that $\mathbb{X}$ and $\mathbb{X}^s$ are* ***strongly equivalent*** *internal categories.*

Proof. We set $\mathbb{X}^s$ to be the $\Sigma$-partial element classifier $\mathbb{X}^+$ itself. The strict algebra structure $\alpha : \mathbb{X}^{++} \longrightarrow \mathbb{X}^+$ is given by the multiplication operation of the partial element classifier monad, *i.e.* the dependent conjunction of the dominance $\Sigma$. The strictness of the algebra structure follows from the fact that $\Sigma$ is univalent, like any dominance. The equivalence $\mathbb{X}^+ \longrightarrow \mathbb{X}$ is given by the pseudo-algebra $\Pi$. □

We therefore combine Lemma 5.7 with Lemma 5.9 to obtain the following theorem.

Theorem 5.10. *Let $\mathscr{E}$ be an elementary topos model of* **iGDTT** *and let $\mathbb{C}$ be a category internal to $\mathbb{V}_0 \in \mathscr{E}$. The category of internal diagrams $[\mathbb{C}, \mathscr{E}]$ is a model of* **iGDTT** *in which:*

(1) *the universes $\mathbb{P}, \mathbb{V}_i$ are modelled by Hofmann–Streicher lifting;*
(2) *the impredicative universe $\mathbb{U} \in \mathbb{V}_i$ is modelled by a universe strongly equivalent to the Hofmann–Streicher lifting of $\mathbb{U} : \mathscr{E}$, but equipped with a strict extension structure for $\mathbb{P}$;*
(3) *the later modality $\blacktriangleright$ is computed pointwise,* i.e. *we have $(\blacktriangleright A)c = \blacktriangleright(Ac)$.*

We will use this result twice: first to extend models of **iGDTT** with constructs for relational reasoning and parametricity (Section 5.3), and second to extend models of **iGDTT** with higher-order store (Section 5.4). The culmination of this process is a standard model for $\mathbf{iGDTT}^{\mathsf{ref}}_{\textsc{lrat}}$ that modularly combines realizability and presheaves.

[8] A similar observation is made by Escardó [2021] in his investigation of injective types in univalent mathematics.

[9] Algebras for the pointed endofunctor $-^+$ play an important role in the recent work of Awodey on Quillen model structures in cubical sets [Awodey 2021], from whom we have borrowed some of our notation.

### 5.3 Logical relations as types in the span model of iGDTT

Prior to the modeling $\mathbf{iGDTT}^{\mathsf{ref}}_{\text{LRAT}}$, we expose the construction of the span model of **iGDTT** on top of a base model. This intermediate model does not possess the necessary structure to interpret references, but it does support the relational primitives of $\mathbf{iGDTT}^{\mathsf{ref}}_{\text{LRAT}}$. We will eventually use **iGDTT** itself as an internal language within this model in Section 5.4 to build the store model.

*5.3.1 The span model.* Given a topos model $\mathscr{E}$ of **iGDTT** we may take the topos $\mathbf{Span}\,\mathscr{E}$ of spans in $\mathscr{E}$, which can be constructed as a category of diagrams $[\mathbb{C},\mathscr{E}]$ where $\mathbb{C}$ is the generic span $\{\mathsf{L} \leftarrow \mathsf{C} \rightarrow \mathsf{R}\}$ viewed as an internal poset in $\mathbb{U} \in \mathscr{E}$. When $\mathscr{E}$ is the category of presheaves on a well-founded poset $\mathbb{O}$ computed in a realizability topos such as **Eff** as in Section 5.1, we obtain from Theorem 5.10 a new model of **iGDTT** in $\mathbf{Span}\,\mathscr{E}$ in which all guarded constructs are computed pointwise. The span model $\mathbf{Span}\,\mathscr{E}$ contains propositions $\mathbf{\Phi}_{\text{L}}, \mathbf{\Phi}_{\text{R}}$ given by the representables $\mathsf{y}_{\mathbb{C}^\circ}\mathsf{L}, \mathsf{y}_{\mathbb{C}^\circ}\mathsf{R}$ respectively. The span model is moreover closed under the *refinement* type connective $[\mathbf{\Phi} \hookrightarrow x : A \mid Bx]$ where $\mathbf{\Phi} = \mathbf{\Phi}_{\text{L}}+\mathbf{\Phi}_{\text{R}}$ is the (disjoint) union as usual; this follows as the subterminals $\mathbf{\Phi}_{\text{L}}, \mathbf{\Phi}_{\text{R}}$ have pointwise decidable image in $\mathscr{E}$ [Orton and Pitts 2016].

*5.3.2 Modal geometry of the span model.* We may further unravel the open and closed modalities of $\mathbf{Span}\,\mathscr{E}$ to provide additional computational and geometric intuition. First we comment that the propositions $\mathbf{\Phi}_{\text{L}}, \mathbf{\Phi}_{\text{R}}, \mathbf{\Phi}$ are concretely realized by the spans $\{1_{\mathscr{E}} \leftarrow \emptyset_{\mathscr{E}} \rightarrow \emptyset_{\mathscr{E}}\}$, $\{\emptyset_{\mathscr{E}} \leftarrow \emptyset_{\mathscr{E}} \rightarrow 1_{\mathscr{E}}\}$, and $\{1_{\mathscr{E}} \leftarrow \emptyset_{\mathscr{E}} \rightarrow 1_{\mathscr{E}}\}$ respectively.

*Characterization of open modalities.* For any subterminal object $\phi \rightarrowtail 1_{\mathbf{Span}\,\mathscr{E}}$, the subcategory of $\mathbf{Span}\,\mathscr{E}$ spanned by $\phi$-transparent objects can be expressed as the slice $\mathbf{Span}\,\mathscr{E} \downarrow \phi$. But in the case of the specific propositions that we have distinguished, more explicit computations are available:

(1) A $\mathbf{\Phi}$-transparent object is one that is isomorphic to a product span $\{X_{\text{L}} \leftarrow X_{\text{L}} \times X_{\text{R}} \rightarrow X_{\text{R}}\}$.
(2) A $\mathbf{\Phi}_{\text{L}}$-transparent object is one that is isomorphic to a digram of the form $\{X_{\text{L}} \leftarrow X_{\text{L}} \rightarrow 1_{\mathscr{E}}\}$.
(3) A $\mathbf{\Phi}_{\text{R}}$-transparent object is one that is isomorphic to a digram of the form $\{1_{\mathscr{E}} \leftarrow X_{\text{R}} \rightarrow X_{\text{R}}\}$.

Thus we compute the open modalities explicitly:

(1) The open modality $(\mathbf{\Phi} \Rightarrow -)$ takes a span $\{X_{\text{L}} \leftarrow \tilde{X} \rightarrow X_{\text{R}}\}$ to the span $\{X_{\text{L}} \leftarrow X_{\text{L}} \times X_{\text{R}} \rightarrow X_{\text{R}}\}$.
(2) The open modality $(\mathbf{\Phi}_{\text{L}} \Rightarrow -)$ takes a span $\{X_{\text{L}} \leftarrow \tilde{X} \rightarrow X_{\text{R}}\}$ to the span $\{X_{\text{L}} \leftarrow X_{\text{L}} \rightarrow 1_{\mathscr{E}}\}$.
(3) The open modality $(\mathbf{\Phi}_{\text{R}} \Rightarrow -)$ takes a span $\{X_{\text{L}} \leftarrow \tilde{X} \rightarrow X_{\text{R}}\}$ to the span $\{1_{\mathscr{E}} \leftarrow X_{\text{R}} \rightarrow X_{\text{R}}\}$.

Based on the above, it is easy to see (*e.g.*) that $\mathbf{\Phi} \Rightarrow A$ is isomorphic to $(\mathbf{\Phi}_{\text{L}} \Rightarrow A) \times (\mathbf{\Phi}_{\text{R}} \Rightarrow A)$.

*Characterization of closed modality.* Here we give an analogous discussion of the closed modality. A $\mathbf{\Phi}$-sealed object is one that is isomorphic to a diagram of the form $\{1_{\mathscr{E}} \leftarrow \tilde{X} \rightarrow 1_{\mathscr{E}}\}$. Then the closed modality $(\mathbf{\Phi} \bullet -)$ takes a span $\{X_{\text{L}} \leftarrow \tilde{X} \rightarrow X_{\text{R}}\}$ to the span $\{1_{\mathscr{E}} \leftarrow \tilde{X} \rightarrow 1_{\mathscr{E}}\}$.

*5.3.3 Synthetic weak bisimulation.* In the span model of **iGDTT**, we may define a new type connective $\widetilde{\mathsf{L}} : \{\mathbb{U} \rightarrow \mathbb{U} \mid \mathbf{\Phi} \hookrightarrow \mathsf{L}\}$ that expresses a weak bisimulation between two computations; in other words, we will have $\mathbf{\Phi} \Rightarrow \widetilde{\mathsf{L}}\,A = \mathsf{L}\,A$ and hence both $\mathbf{\Phi}_{\text{L}} \Rightarrow \widetilde{\mathsf{L}}\,A = \mathsf{L}\,A$ and $\mathbf{\Phi}_{\text{R}} \Rightarrow \widetilde{\mathsf{L}}\,A = \mathsf{L}\,A$. Given $u_{\text{L}} : \mathbf{\Phi}_{\text{L}} \Rightarrow \mathsf{L}\,A$ and $u_{\text{R}} : \mathbf{\Phi}_{\text{R}} \Rightarrow \mathsf{L}\,A$, an element of $\{\widetilde{\mathsf{L}}\,A \mid \mathbf{\Phi}_{\text{L}} \hookrightarrow u_{\text{L}}, \mathbf{\Phi}_{\text{R}} \hookrightarrow u_{\text{R}}\}$ will be a proof that $u_{\text{L}}$ and $u_{\text{R}}$ are weakly bisimilar. Because $A$ itself can be any type (*e.g.* any synthetic correspondence), our definition of $\widetilde{\mathsf{L}}\,A$ can be seen to be an adaptation of the lifting of a correspondence from

Møgelberg and Paviotti [2016]. We define $\widetilde{\mathsf{L}}A$ by solving a guarded recursive domain equation in $\mathbb{U}$:

$$
\begin{array}{l}
\widetilde{\mathsf{L}}A \triangleq [\Phi \hookrightarrow u : \mathsf{L}A \mid \mathsf{WeaklyBisimilar}_A u] \\
\\
\textbf{data } \mathsf{Done}_A : (\Phi \Rightarrow \mathsf{L}A) \to \mathbb{U} \textbf{ where} \\
\quad \mathsf{stop} : (a : \Phi \Rightarrow A) \to \mathsf{Done}_A(\eta a) \\
\quad \mathsf{waitR} : (a : \Phi \Rightarrow A, n : \mathbb{N}_{\geq 1}) \to \mathsf{Done}_A\,[\Phi_{\mathrm{L}} \hookrightarrow \eta a, \Phi_{\mathrm{R}} \hookrightarrow \delta^n \eta a] \\
\quad \mathsf{waitL} : (a : \Phi \Rightarrow A, n : \mathbb{N}_{\geq 1}) \to \mathsf{Done}_A\,[\Phi_{\mathrm{L}} \hookrightarrow \delta^n \eta a, \Phi_{\mathrm{R}} \hookrightarrow \eta a] \\
\\
\textbf{quotient data } \mathsf{WeaklyBisimilar}_A : (\Phi \Rightarrow \mathsf{L}A) \to \mathbb{U} \textbf{ where} \\
\quad \mathsf{done} : \{u : \Phi \Rightarrow \mathsf{L}A\} \to \mathsf{Done}_A u \to \mathsf{WeaklyBisimilar}_A u \\
\quad \mathsf{step} : (u : \blacktriangleright \widetilde{\mathsf{L}}A) \to \mathsf{WeaklyBisimilar}_A\,(\vartheta_{\mathsf{L}A}\, u) \\
\quad \star : \{\_ : \Phi, u : \mathsf{L}A\} \to \mathsf{WeaklyBisimilar}_A u \\
\quad \_ : \{\_ : \Phi, u : \mathsf{L}A, p : \mathsf{WeaklyBisimilar}_A u\} \to p = \star
\end{array}
$$

We have ensured that $\mathsf{WeaklyBisimilar}_A$ is valued in $\Phi$-sealed types by defining it as a *quotient inductive* definition; here again we recall Remark 4.6. Thus the use of the refinement type in $\widetilde{\mathsf{L}}A$ is well-defined. We note that $\widetilde{\mathsf{L}}A$ enjoys a similar interface to that of $\mathsf{L}A$; in particular, it inherits the structure of a guarded domain from its components, meaning that it supports recursion:

$$
\begin{array}{l}
\Phi \hookrightarrow \vartheta_{\widetilde{\mathsf{L}}A} u \triangleq \vartheta_{\mathsf{L}A}\, u \\
\underline{\vartheta_{\widetilde{\mathsf{L}}A} u} \triangleq \mathsf{step}\,(\mathsf{next}[x \leftarrow u].\, x)
\end{array}
$$

Likewise, we may define a unit map $\widetilde{\eta} : A \to \widetilde{\mathsf{L}}A$ as follows:

$$
\begin{array}{l}
\Phi \hookrightarrow \widetilde{\eta}\, a \triangleq \eta\, a \\
\underline{\widetilde{\eta}\, a} \triangleq \mathsf{done}\,(\mathsf{stop}\, a)
\end{array}
$$

In order to define the Kleisli extension for $\widetilde{\mathsf{L}}A$ we must first define an operation that adds a step on the left or on the right; we define the left-handed stepper $\delta_{\mathrm{L}}$ below, and treat $\delta_{\mathrm{R}}$ symmetrically.

$$
\begin{array}{l}
\delta_{\mathrm{L}} : \widetilde{\mathsf{L}}A \to \widetilde{\mathsf{L}}A \\
\Phi \hookrightarrow \delta_{\mathrm{L}} u \triangleq [\Phi_{\mathrm{L}} \hookrightarrow \delta u, \Phi_{\mathrm{R}} \hookrightarrow u] \\
\underline{\delta_{\mathrm{L}} u} \triangleq \textbf{case } \underline{u} \textbf{ of} \\
\qquad \mathsf{done}\,(\mathsf{stop}\, a) \hookrightarrow \mathsf{done}\,(\mathsf{waitL}\, 1\, a) \\
\qquad \mathsf{done}\,(\mathsf{waitL}\, a\, n) \hookrightarrow \mathsf{done}\,(\mathsf{waitL}\,(n+1)\, a) \\
\qquad \mathsf{done}\,(\mathsf{waitR}\, a\, n) \hookrightarrow \mathsf{step}\,(\mathsf{next}\,(\mathsf{waitR}_?\,(n-1)\, a)) \\
\qquad \mathsf{step}\, u' \hookrightarrow \mathsf{step}\,(\mathsf{next}[x \leftarrow u'].\, \delta_{\mathrm{L}} x) \\
\qquad \star \hookrightarrow [\Phi_{\mathrm{L}} \hookrightarrow \delta u, \Phi_{\mathrm{R}} \hookrightarrow u] \\
\\
\mathsf{waitR}_? : (a : \Phi \Rightarrow A)\,(n : \mathbb{N}) \to \mathsf{Done}_A\,[\Phi_{\mathrm{L}} \hookrightarrow \eta a, \Phi_{\mathrm{R}} \hookrightarrow \delta^n \eta a] \\
\mathsf{waitR}_?\, a\, 0 \triangleq \mathsf{stop}\, a \\
\mathsf{waitR}_?\, a\,(n \geq 1) \triangleq \mathsf{waitR}\, a\, n
\end{array}
$$

To see that the case analysis in $\underline{\delta_{\mathrm{L}} u}$ is well-defined as a map out of the quotient, it suffices to observe that all clauses return $\delta u$ under $\Phi_{\mathrm{L}}$ and $u$ under $\Phi_{\mathrm{R}}$. Next we define $\mathsf{step}_{\mathrm{L}}, \mathsf{step}_{\mathrm{R}}$ to be the generic effects $\delta_{\mathrm{L}}(\mathsf{ret}\,())$ and $\delta_{\mathrm{R}}(\mathsf{ret}\,())$ respectively. We finally define the Kleisli extension for $\widetilde{\mathsf{L}}$ as well, extending the Kleisli extension for $\mathsf{L}$:

$$\begin{array}{ll}
\begin{array}{l}
\Phi \hookrightarrow \widetilde{\mathsf{bind}}\,u\,f \triangleq \mathsf{bind}\,u\,f \\
\underline{\widetilde{\mathsf{bind}}\,u\,f} \triangleq \mathbf{case}\,\underline{u}\,\mathbf{of} \\
\qquad\qquad \mathsf{done}\,d \hookrightarrow \mathsf{finalize}\,d\,f \\
\qquad\qquad \mathsf{step}\,u' \hookrightarrow \mathsf{step}\,(\mathsf{next}[x \leftarrow u'].\,\widetilde{\mathsf{bind}}\,x\,f) \\
\qquad\qquad \star \hookrightarrow \mathsf{bind}\,u\,f
\end{array}
&
\begin{array}{l}
\mathsf{finalize}\,(\mathsf{stop}\,a)\,f \triangleq f\,a \\
\mathsf{finalize}\,(\mathsf{waitL}\,n\,a)\,f \triangleq \delta^n_{\mathrm{L}}\,(f\,a) \\
\mathsf{finalize}\,(\mathsf{waitR}\,n\,a)\,f \triangleq \delta^n_{\mathrm{R}}\,(f\,a)
\end{array}
\end{array}$$

### 5.4 Higher-order store in the possible worlds model of $\mathbf{iGDTT}^{\mathsf{ref}}_{\mathrm{LRAT}}$

We recall the construction of the category $\mathbb{W}$ internal to $\mathbb{V}_0$ as well as the objects of heaps $\mathbb{H}_w$ from Section 2.4.1. In this section, we apply Theorem 5.10 to $[\mathbb{W}, \mathbf{Span}\,\mathscr{E}]$ to construct a dependently typed version of the store model. In our previous model, the collection of types was global and did not depend on Kripke worlds; in the new version, the types themselves depend on heap shapes. We define the reference type as follows, writing $B_{|w'} : [w' \downarrow \mathbb{W}, \mathbb{U}]$ for the obvious restriction of $B : [\mathbb{W}, \mathbb{U}]$ to the coslice:

$$\begin{array}{l}
A : \overline{\mathbb{U}} \vdash \mathsf{ref}\,A : \overline{\mathbb{U}} \\
\mathsf{ref}_w(A : [w \downarrow \mathbb{W}, \mathbb{U}])\,(w' \geq w) \triangleq \{i \in |w'| \mid \blacktriangleright[B \leftarrow w'i].B_{|w'} = A_{|w'}\}
\end{array}$$

We define the internal state monad and its $\blacktriangleright$-algebra structure as follows, using the weak bisimulation monad $\widetilde{\mathsf{L}}$ pointwise:

$$\begin{array}{ll}
\begin{array}{l}
A : \overline{\mathbb{U}} \vdash \mathsf{T}\,A : \overline{\mathbb{U}} \\
\mathsf{T}_w A w_1 \triangleq \forall_{w_2 \geq w_1} \mathbb{H}_{w_2} \to \widetilde{\mathsf{L}}\,\exists_{w_3 \geq w_2} \mathbb{H}_{w_3} \times A w_3
\end{array}
&
\begin{array}{l}
A : \overline{\mathbb{U}} \vdash \vartheta_{\mathsf{T}\,A} : \blacktriangleright \mathsf{T}\,A \to \mathsf{T}\,A \\
\vartheta_{w,\mathsf{T}_w A} m w' h \triangleq \vartheta\,(\mathsf{next}[u \leftarrow m].u w' h)
\end{array}
\end{array}$$

$$\begin{array}{ll}
\begin{array}{l}
A : \overline{\mathbb{U}} \vdash \mathsf{ret}_A : A \to \mathsf{T}\,A \\
\mathsf{ret}_{w,A} a w' h \triangleq \eta\,(\mathsf{pack}\,(w', (h, a_{|w_2})))
\end{array}
&
\begin{array}{l}
A, B : \overline{\mathbb{U}} \vdash \mathsf{bind}_{A,B} : \mathsf{T}\,A \times (A \to \mathsf{T}\,B) \to \mathsf{T}\,B \\
\mathsf{bind}_{w,A,B}(m,k) w' h = \mathsf{pack}\,(w'', (h', a)) \leftarrow m w h; k_{w''} a w'' h'
\end{array}
\end{array}$$

The steppers $\mathsf{step}, \mathsf{step}_{\mathrm{L}}, \mathsf{step}_{\mathrm{R}} : \mathsf{T}\,1$ are defined *pointwise* using the corresponding constructs of $\widetilde{\mathsf{L}}$. We define the generic effects for the getter and setter below:

$$\begin{array}{ll}
\begin{array}{l}
A : \overline{\mathbb{U}} \vdash \mathsf{get}_A : \mathsf{ref}\,A \to \mathsf{T}\,A \\
\mathsf{get}_{w,A} l w' h \triangleq \\
\quad x \leftarrow \vartheta\,(\mathsf{next}[z \leftarrow hl].\eta z); \\
\quad \eta\,(\mathsf{pack}\,(w', (h, x)))
\end{array}
&
\begin{array}{l}
A : \overline{\mathbb{U}} \vdash \mathsf{set}_A : \mathsf{ref}\,A \times A \to \mathsf{T}\,1 \\
\mathsf{set}_{w,A}(l, x) w' h \triangleq \\
\quad \eta\,(\mathsf{pack}\,(w', (h[l \mapsto x_{|w'}], *)))
\end{array}
\end{array}$$

The allocator is the only subtle part; it is here that we will need to use the strict extension structure that we have assumed on $\mathbb{U}$ in $\mathscr{E}$.

$$\begin{array}{l}
A : \overline{\mathbb{U}} \vdash \mathsf{new}_A : A \to \mathsf{T}\,(\mathsf{ref}\,A) \\
\mathsf{new}_{w,A} x w' h \triangleq \\
\quad \mathbf{let}\ i = \mathsf{fresh}\,|w'|; \\
\quad \mathbf{let}\ w'' = w' \cup \{i \mapsto \mathsf{next}\,(\lambda w_0.(w_0 \geq w')_*(A w_0))\}; \\
\quad \mathbf{let}\ h' = h_{|w''} \cup \{i \mapsto \mathsf{next}\,x_{|w''}\}; \\
\quad \eta\,(\mathsf{pack}\,(w'', (i, h')))
\end{array}$$

The problem solved by strict extension above is the following: a type $A : \overline{\mathbb{U}}$ at world $w$ has extent only on the coslice $w \downarrow \mathbb{W}$, and yet to extend the world by a new cell we must provide a type that has extent on all of $\mathbb{W}$. We use strict extension to extend $A$ to a global type in a way that agrees *strictly* with $A$ after $w' \geq w$. We synthesize the preceding discussion into the following result:

THEOREM 5.11. *Let $\mathscr{E}$ be an elementary topos model of* **iGDTT**. *The lifting of this model to $[\mathbb{W}, \mathbf{Span}\,\mathscr{E}]$ extends to a model of* $\mathbf{iGDTT}^{\mathsf{ref}}_{\mathrm{LRAT}}$.

Proof. Applying Theorem 5.10 we obtain a model of **iGDTT**. The relational constructs of $\mathbf{iGDTT}^{\mathsf{ref}}_{\mathrm{LRAT}}$ ($\mathbf{\Phi}_{\mathrm{L}}$, $\mathbf{\Phi}_{\mathrm{R}}$, *etc.*) are lifted pointwise from $\mathbf{Span}\,\mathscr{E}$. The remaining constructs—those governing state—are interpreted using the above definitions of $\mathsf{T}$, new, get, and set. □

Recalling that $\mathbf{iGDTT}^{\mathsf{ref}}$ embeds into $\mathbf{iGDTT}^{\mathsf{ref}}_{\mathrm{LRAT}}$, we obtain the following by the same reasoning as Corollary 5.5:

Corollary 5.12. *Both* $\mathbf{iGDTT}^{\mathsf{ref}}$ *and* $\mathbf{iGDTT}^{\mathsf{ref}}_{\mathrm{LRAT}}$ *are consistent.*

Fix a closed term $M : \mathsf{T}\,\mathbb{N}$. Inspecting the interpretation of $M$ within $[\mathbb{W}, \mathbf{Span}\,\mathscr{E}]$, we see that it corresponds to a global element of $[\![M]\!] : 1 \longrightarrow \mathsf{T}\,\mathbb{N}$ in $[\mathbb{W}, \mathbf{Span}\,\mathscr{E}]$. As $\mathbb{W}$ possesses an initial object (the empty heap), $[\![M]\!]$ is determined by its instantiation $[\![M]\!]_{\emptyset} : 1 \longrightarrow \mathsf{T}_{\emptyset}\mathbb{N}$.

Given any semantic world $w$ and heap $h : \mathbb{H}_w$, we therefore may run $t_{\emptyset}$ to obtain an element $\widetilde{\mathsf{L}}\,\mathbb{N}$ within $\mathbf{Span}\,\mathscr{E}$. We present this using **iGDTT** as the internal language of $\mathbf{Span}\,\mathscr{E}$:

$$\begin{aligned}&\mathsf{run} : (w : \mathbb{W}) \to \mathbb{H}_w \to \widetilde{\mathsf{L}}\,\mathbb{N}\\ &\mathsf{run}_w\ h\ t = (w', h', n) \leftarrow t_{\emptyset}\ w\ h; \eta(n)\end{aligned}$$

Denote the restriction of $[\![M]\!]$ by $\mathbf{\Phi}_{\mathrm{L}}$ (resp. $\mathbf{\Phi}_{\mathrm{R}}$) by $[\![M]\!]_{\mathrm{L}}$ (resp. $[\![M]\!]_{\mathrm{R}}$). The preceding discussion substantiates the claim made in Section 4.3:

Theorem 5.13. *Fix a closed term* $M : \mathsf{T}\,\mathbb{N}$ *in* $\mathbf{iGDTT}^{\mathsf{ref}}_{\mathrm{LRAT}}$. *For any* $w : \mathbb{W}$ *and a heap* $h : \mathbb{H}_w$, *there is an element of* $\mathsf{WeaklyBisimilar}_{\mathbb{N}}[\mathbf{\Phi}_L \hookrightarrow \mathsf{run}_w\ h\ [\![M]\!]_L, \mathbf{\Phi}_R \hookrightarrow \mathsf{run}_w\ h\ [\![M]\!]_R]$ i.e. *a weak bisimulation between the results of executing* $[\![M]\!]_L$ *and* $[\![M]\!]_R$.

## 6 CALL-BY-PUSH-VALUE DECOMPOSITION OF POLYMORPHISM & STATE

The model of higher-order store in $\mathbf{iGDTT}^{\mathsf{ref}}$ that we constructed in Section 3 is a special case of a much more general construction on models of *polymorphic call-by-push-value* that we describe here. The results of the current section allow one to incorporate additional computational effects modularly in the spirit of algebraic effects.

We defer a formal presentation of the syntax of polymorphic call-by-push-value to Appendix A; but for intuition, we assume the following judgmental structure:

(1) $\Psi \vdash$ means that $\Psi$ is a type context.
(2) $\Psi \vdash A$ *vtype* means that $A$ is a value type in context $\Psi$.
(3) $\Psi \vdash X$ *ctype* means that $X$ is a computation type in context $\Psi$.
(4) $\Psi; \Gamma \vdash$ means that $\Gamma$ is a value context in type context $\Psi$.
(5) $\Psi; \Gamma \vdash V : A$ means that $V$ is a value of type $A$ in context $\Psi; \Gamma$.
(6) $\Psi; \Gamma \mid X \vdash K : Y$ means that $K$ is a *stack* from $X$ and $Y$ in context $\Psi; \Gamma$.
(7) $\Psi; \Gamma \vdash M : X$ means that $M$ is computation of type $X$ in context $\Psi; \Gamma$.

The results of this section will be stated at the level of categorical semantics rather than syntax. In paritcular, we define a categorical notion of model for polymorphic call-by-push-value inspired by that of Vákár [2017], generalizing Levy's adjunction models [Levy 2003b] to account for the parameterization of types over kinds.

*Definition 6.1.* A ***fibered cbpv model*** is given by the following data:

(1) a category $\mathscr{B}$ of *kinds* equipped with finite products,
(2) a fibered category $\mathscr{C}$ over $\mathscr{B}$ of *value types* and *values*, with a fibered terminal object
(3) a fibered category $\mathscr{D}$ over $\mathscr{B}$ of *computation types* and *stacks*
(4) a full comprehension structure $\{-\} : \mathscr{C} \hookrightarrow \mathbf{P}_{\mathscr{B}}$,
(5) a fibered functor $\mathsf{U} : \mathscr{D} \longrightarrow \mathscr{C}$ over $\mathscr{B}$ with a fibered left adjoint $\mathsf{F} \dashv \mathsf{U}$.

In other words, we require the following fibered structure over $\mathscr{B}$ depicted in Diagram 1 below; note that $\mathbf{P}_{\mathscr{B}}$ may be only a *displayed* rather than *fibered* category unless $\mathscr{B}$ has pullbacks, but this does not prevent stating the conditions of a fully faithful displayed functor $\{-\} : \mathscr{C} \hookrightarrow \mathbf{P}_{\mathscr{B}}$.

$$\begin{tikzcd}
\mathscr{D} \arrow[r, bend right, "\mathsf{U}"'] \arrow[r, phantom, "\perp"] & \mathscr{C} \arrow[l, bend right, "\mathsf{F}"'] \arrow[r, hook, "\{-\}"] & \mathbf{P}_{\mathscr{B}}
\end{tikzcd} \tag{1}$$

A fibered cbpv model is a categorical semantics for the judgments and adjunctives of polymorphic cbpv in the following way:

(1) a type context $\Psi \vdash$ denotes an object of $\mathscr{B}$,
(2) a value context $\Psi; \Gamma \vdash$ denotes an object of $\mathscr{C}_\Psi$,
(3) a value type $\Psi \vdash A$ *vtype* denotes an object of $\mathscr{C}_\Psi$,
(4) a computation type $\Psi \vdash X$ *ctype* denotes an object of $\mathscr{D}_\Psi$,
(5) a value $\Psi; \Gamma \vdash V : A$ denotes a morphism $\Gamma \longrightarrow A$ in $\mathscr{C}_\Psi$,
(6) a stack $\Psi; \Gamma \mid X \vdash K : Y$ denotes a morphism $\pi_\Gamma^* X \longrightarrow \pi_\Gamma^* Y$ in $\mathscr{D}_{\{\Gamma\}_\Psi}$, where $\pi_\Gamma : \{\Gamma\}_\Psi \longrightarrow \Psi \in \mathscr{B}$ is induced by the comprehension structure,
(7) a computation $\Psi; \Gamma \vdash M : X$ denotes a morphism $\mathsf{F}\,\Gamma \longrightarrow X$ in $\mathscr{D}_\Psi$ or, equivalently, a morphism $\Gamma \longrightarrow \mathsf{U}\,X$ in $\mathscr{C}_\Psi$.

*Example 6.2.* The ordinary adjunction models of cbpv are a special case of the above. Let $\underline{\mathcal{S}}$ be a locally $\mathscr{V}$-indexed category. Then let $\mathscr{B}$ be $\mathscr{V}$, let $\mathscr{C}$ be the simple fibration of $\mathscr{V}$, and let $\mathscr{D}_A$ have the same objects as $\underline{\mathcal{S}}$ such that a morphism $X \longrightarrow Y$ in $\mathscr{D}_A$ is given by an element of $\underline{\mathcal{S}}_A(X, Y)$.

*Definition 6.3.* A ***polymorphic fibered cbpv model*** is a fibered cbpv model such that $\mathscr{C}$ is equipped with a weak generic object and $\mathscr{D}$ has simple products in the sense of Jacobs [1999]. We will write $\mathsf{vtp} \in \mathscr{B}$ for the base of the generic object and $\mathsf{val} \in \mathscr{C}_{\mathsf{vtp}}$ for its fiber.

In a polymorphic fibered cbpv model, the generic object represents the type context extension $\Psi, \alpha \vdash$ and the simple products implement the polymorphic types $\forall_\alpha X[\alpha]$.

### 6.1 Preliminaries on fibered categories

Let $\mathscr{B}$ be a category with finite limits and let $\mathbb{C}$ be an internal category in $\mathscr{B}$; let $\mathscr{E}$ be a fibered category over $\mathscr{B}$. There is a category $[\mathbb{C}, \mathscr{E}]_{\mathscr{B}}$ over *internal diagrams* over $\mathscr{B}$. An internal diagram $F \in [\mathbb{C}, \mathscr{E}]_{\mathscr{B}}$ is given by a displayed object $\mathsf{ob}_F \in \mathscr{E}_{\mathsf{ob}_{\mathbb{C}}}$ and a vertical morphism $\mathsf{hom}_F : \partial_0^* \mathsf{ob}_F \longrightarrow \partial_1^* \mathsf{ob}_F \in \mathscr{E}_{\mathsf{hom}_{\mathbb{C}}}$ satisfying internal versions of the functor laws.

In the special case of the fundamental fibered category $\mathbf{P}_{\mathscr{B}}$ over $\mathscr{B}$, an element of $[\mathbb{C}, \mathbf{P}_{\mathscr{B}}]_{\mathscr{B}}$ is the same as an *internal diagram* in $\mathscr{B}$ in the sense of Borceux [1994]. Such an internal diagram $F \in [\mathbb{C}, \mathbf{P}_{\mathscr{B}}]_{\mathscr{B}}$ gives rise to an *internal total category* $\mathbb{C} \ltimes F \in \mathscr{B}$, whose object of objects $\mathsf{ob}_{\mathbb{C} \ltimes F}$ is the sum $\coprod_{\mathsf{ob}_{\mathbb{C}}} \mathsf{ob}_F$ where $\mathsf{ob}_F \in \mathscr{B} \downarrow \mathsf{ob}_{\mathbb{C}}$ is the displayed object associated to $F$.

### 6.2 The polymorphic store model construction

Let $\mathscr{B}$ be a locally cartesian closed category (so that $\mathbf{P}_{\mathscr{B}}$ is a locally small fibered category) and let $(\mathscr{C}, \mathscr{D}, \{-\}, \mathsf{U}, \mathsf{F})$ be a polymorphic fibered cbpv model over $\mathscr{B}$ such that $\mathscr{C}$ has both products and coproducts over $\mathscr{B}$ in the sense of Jacobs [1999, Definition 1.9.4].

Let $\mathbb{W}, \mathbb{H}$ be internal categories in $\mathscr{B}$ such that $\mathbb{H}$ is discrete, and let $s : \mathbb{H} \longrightarrow \mathsf{ob}_{\mathbb{W}}$ be a function. We may define a new *stateful* polymorphic fibered cbpv model indexed in worlds drawn from $\mathbb{W}$ with states in $\mathbb{H}$. We define $\overline{\mathscr{B}}$ to be the category of internal diagrams $[\mathbb{W}, \mathbf{P}_{\mathscr{B}}]_{\mathscr{B}}$.

We will define a fibered category $\overline{\mathscr{C}}$ over $\overline{\mathscr{B}}$.

(1) For $\Psi \in \overline{\mathscr{B}}$, a displayed object $A \in \overline{\mathscr{C}}_{\Psi}$ is given by an internal diagram $A \in [\mathbb{W} \ltimes \Psi, \mathscr{C}]_{\mathscr{B}}$.
(2) For $\psi : \Phi \longrightarrow \Psi \in \overline{\mathscr{B}}$, a displayed morphism $f : A \longrightarrow B$ in $\overline{\mathscr{C}}$ over $\psi$ is given by a morphism $f : \mathbb{W} \ltimes \psi \cdot A \longrightarrow B$ in $[\mathbb{W} \ltimes \Psi, \mathscr{C}]_{\mathscr{B}}$.

We define $\overline{\mathscr{D}}$ over $\overline{\mathscr{B}}$ in a similar fashion to the above, flipping the variance.

(1) For $\Psi \in \overline{\mathscr{B}}$, a displayed object $X \in \overline{\mathscr{D}}_{\Psi}$ is given by an internal diagram $X \in \left[(\mathbb{W} \ltimes \Psi)^{\circ}, \mathscr{D}\right]_{\mathscr{B}}$.
(2) For $\psi : \Phi \longrightarrow \Psi \in \overline{\mathscr{B}}$, a displayed morphism $k : X \to Y$ over $\psi$ is given by a morphism $k : A \longrightarrow \mathbb{W} \ltimes \psi^{*}B$ in $\left[(\mathbb{W} \ltimes \Phi)^{\circ}, \mathscr{D}\right]_{\mathscr{B}}$.

*Construction 6.4 (Comprehension structure).* We define the comprehension $\overline{\{-\}} : \overline{\mathscr{C}} \hookrightarrow \mathbf{P}_{\overline{\mathscr{B}}}$ by applying the existing comprehension structure $\{-\} : \mathscr{C} \hookrightarrow \mathbf{P}_{\mathscr{B}}$ pointwise.

*Construction 6.5 (The lifted store span).* The basic notion of store $\mathbb{H}$ is a discrete internal $\mathscr{B}$-category indexed in the internal category of worlds $\mathbb{W}$; in the fibered store model, we will treat the total internal category $\mathbb{W} \ltimes \Psi$ of $\Psi \in [\mathbb{W}, \mathbf{P}_{\mathscr{B}}]_{\mathscr{B}}$ as an internal category of worlds and thus we must define the corresponding notion of store $\mathbb{H}_{\Psi}$. We achieve this by pullback as follows:

$$\begin{tikzcd}
\mathbb{W} \ltimes \Psi \arrow[d, "p_{\Psi}"'] & \mathbb{H}_{\Psi} \arrow[l, "s_{\Psi}"'] \arrow[r, "s^{\circ}_{\Psi}"] \arrow[d] \arrow[dl, phantom, "\llcorner", very near start] \arrow[dr, phantom, "\lrcorner", very near start] & (\mathbb{W} \ltimes \Psi)^{\circ} \arrow[d, "p^{\circ}_{\Psi}"] \\
\mathbb{W} & \mathbb{H} \arrow[l, "s"] \arrow[r, "s^{\circ}"'] & \mathbb{W}^{\circ}
\end{tikzcd}$$

*Construction 6.6 (The left adjoint).* We may define a functor $\overline{\mathsf{F}} : \overline{\mathscr{C}} \longrightarrow \overline{\mathscr{D}}$ over $\overline{\mathscr{B}}$ using the fact that $\mathscr{C}$ has $\mathscr{B}$-coproducts; the component $\overline{\mathsf{F}}_{\Psi}$ is the following composite:

$$[\mathbb{W} \ltimes \Psi, \mathscr{C}]_{\mathscr{B}} \xrightarrow{\Delta_{s_{\Psi}}} [\mathbb{H}_{\Psi}, \mathscr{C}]_{\mathscr{B}} \xrightarrow{\coprod_{s^{\circ}_{\Psi}}} \left[(\mathbb{W} \ltimes \Psi)^{\circ}, \mathscr{C}\right]_{\mathscr{B}} \xrightarrow{\left[\mathbb{W} \ltimes \Psi^{\circ}, \mathsf{F}\right]_{\mathscr{B}}} \left[(\mathbb{W} \ltimes \Psi)^{\circ}, \mathscr{D}\right]_{\mathscr{B}}$$

Lemma 6.7. *The functor $\overline{\mathsf{F}} : \overline{\mathscr{C}} \longrightarrow \overline{\mathscr{D}}$ is fibered over $\overline{\mathscr{B}}$.*

Proof. Explicitly, for any $f : \Phi \longrightarrow \Psi \in \overline{\mathscr{B}}$ we need for the canonical map $f^{*}\overline{\mathsf{F}}_{\Psi} \longrightarrow \overline{\mathsf{F}}_{\Phi}f^{*}$ to be an isomorphism. Because the base change $f^{*}$ acts by precomposition, this can be checked with an explicit computation starting from a diagram $A \in [\mathbb{W} \ltimes \Psi, \mathscr{C}]_{\mathscr{B}}$; we will use internal notations to facilitate our computation:

$$\begin{aligned}
f^{*}\overline{\mathsf{F}}_{\Psi}A &= f^{*}\left[(\mathbb{W} \ltimes \Psi)^{\circ}, \mathsf{F}\right]_{\mathscr{B}} \textstyle\coprod_{s^{\circ}_{\Psi}} \Delta_{s_{\Psi}} [w, \psi \mapsto Aw\psi] \\
&= f^{*}\left[(\mathbb{W} \ltimes \Psi)^{\circ}, \mathsf{F}\right]_{\mathscr{B}} \textstyle\coprod_{s^{\circ}_{\Psi}} [w, (\psi, h) \mapsto Aw\psi] \\
&= f^{*}\left[(\mathbb{W} \ltimes \Psi)^{\circ}, \mathsf{F}\right]_{\mathscr{B}}\left[w, \psi \mapsto \textstyle\coprod_{w' \geq w} \coprod_{h \in \mathbb{H}_{w'}} Aw'\psi_{|w'}\right] \\
&= f^{*}\left[w, \psi \mapsto \mathsf{F}\left(\textstyle\coprod_{w' \geq w} \coprod_{h \in \mathbb{H}_{w'}} Aw'\psi_{|w'}\right)\right] \\
&= \left[w, \phi \mapsto \mathsf{F}\left(\textstyle\coprod_{w' \geq w} \coprod_{h \in \mathbb{H}_{w'}} Aw'(f_{w}\phi)_{|w'}\right)\right] \\
&= \left[(\mathbb{W} \ltimes \Phi)^{\circ}, \mathsf{F}\right]_{\mathscr{B}}\left[w, \phi \mapsto \textstyle\coprod_{w' \geq w} \coprod_{h \in \mathbb{H}_{w'}} Aw'(f_{w}\phi)_{|w'}\right] \\
&= \left[(\mathbb{W} \ltimes \Phi)^{\circ}, \mathsf{F}\right]_{\mathscr{B}} \textstyle\coprod_{s^{\circ}_{\Phi}} [w, (\phi, h) \mapsto Aw(f_{w}\phi)] \\
&= \left[(\mathbb{W} \ltimes \Phi)^{\circ}, \mathsf{F}\right]_{\mathscr{B}} \textstyle\coprod_{s^{\circ}_{\Phi}} \Delta_{s_{\Phi}} [w, \phi \mapsto Aw(f_{w}\phi)] \\
&= \left[(\mathbb{W} \ltimes \Phi)^{\circ}, \mathsf{F}\right]_{\mathscr{B}} \textstyle\coprod_{s^{\circ}_{\Phi}} \Delta_{s_{\Phi}} f^{*} [w, \psi \mapsto Aw\psi]
\end{aligned}$$

$$= \overline{\mathsf{F}}_\Phi f^* A \qquad \square$$

*Construction 6.8 (The right adjoint).* We define a right adjoint $\overline{\mathsf{U}} : \overline{\mathscr{D}} \longrightarrow \overline{\mathscr{C}}$ using the fact that $\mathscr{C}$ has $\mathscr{B}$-products; the component $\overline{\mathsf{U}}_\Psi$ is the following composite:

$$\left[(\mathbb{W} \ltimes \Psi)^\circ, \mathscr{D}\right]_{\mathscr{B}} \xrightarrow{\left[(\mathbb{W} \ltimes \Psi)^\circ, \mathsf{U}\right]_{\mathscr{B}}} \left[(\mathbb{W} \ltimes \Psi)^\circ, \mathscr{C}\right]_{\mathscr{B}} \xrightarrow{\Delta_{s_\Psi^\circ}} [\mathbb{H}_\Psi, \mathscr{C}]_{\mathscr{B}} \xrightarrow{\prod_{s_\Psi}} [\mathbb{W} \ltimes \Psi, \mathscr{C}]_{\mathscr{B}}$$

Lemma 6.9. *The functor $\overline{\mathsf{U}} : \overline{\mathscr{D}} \longrightarrow \overline{\mathscr{C}}$ is fibered over $\overline{\mathscr{B}}$.*

Proof. Fix $X \in \left[(\mathbb{W} \ltimes \Psi)^\circ, \mathscr{D}\right]_{\mathscr{B}}$ and $f : \Phi \longrightarrow \Psi$.

$$\begin{aligned}
f^*\overline{\mathsf{U}}_\Psi X &= f^* \textstyle\prod_{s_\Psi} \Delta_{s_\Psi^\circ} \left[(\mathbb{W} \ltimes \Psi)^\circ, \mathsf{U}\right]_{\mathscr{B}} [w, \psi \mapsto Xw\psi] \\
&= f^* \textstyle\prod_{s_\Psi} \Delta_{s_\Psi^\circ} [w, \psi \mapsto \mathsf{U}\,(Xw\psi)] \\
&= f^* \textstyle\prod_{s_\Psi} [w, (\psi, h) \mapsto \mathsf{U}\,(Xw\psi)] \\
&= f^* \left[w, \psi \mapsto \textstyle\prod_{w' \geq w} \prod_{h \in \mathbb{H}_{w'}} \mathsf{U}\,(Xw'(\psi_{|w'}))\right] \\
&= \left[w, \phi \mapsto \textstyle\prod_{w' \geq w} \prod_{h \in \mathbb{H}_{w'}} \mathsf{U}\,(Xw'((f_w\phi)_{|w'}))\right] \\
&= \textstyle\prod_{s_\Phi} [w, (\phi, h) \mapsto \mathsf{U}\,(Xw(f_w\phi))] \\
&= \textstyle\prod_{s_\Phi} \Delta_{s_\Phi^\circ} [w, \phi \mapsto \mathsf{U}\,(Xw(f_w\phi))] \\
&= \textstyle\prod_{s_\Phi} \Delta_{s_\Phi^\circ} \left[(\mathbb{W} \ltimes \Phi)^\circ, \mathsf{U}\right]_{\mathscr{B}} [w, \phi \mapsto Xw(f_w\phi)] \\
&= \textstyle\prod_{s_\Phi} \Delta_{s_\Phi^\circ} \left[(\mathbb{W} \ltimes \Phi)^\circ, \mathsf{U}\right]_{\mathscr{B}} f^* [w, \phi \mapsto Xw\phi] \\
&= \overline{\mathsf{U}}_\Phi f^* X \qquad \square
\end{aligned}$$

*Construction 6.10 (The generic object).* The weak generic object of $\overline{\mathscr{C}}$ is built from the weak generic object of $\mathscr{C}$ using the Hofmann–Streicher construction. If $\mathscr{C}$ is split, then $\overline{\mathscr{C}}$ is likewise split and the Hofmann–Streicher construction yields a split generic object.

Lemma 6.11. *The fibered category $\overline{\mathscr{D}}$ has simple products over $\overline{\mathscr{B}}$.*

Corollary 6.12. *The data $(\overline{\mathscr{C}}, \overline{\mathscr{D}}, \overline{\{-\}}, \overline{\mathsf{U}}, \overline{\mathsf{F}})$ consistute a polymorphic fibered cbpv model over $\overline{\mathscr{B}}$.*

*Example 6.13 (Instantiation of the polymorphic store construction).* Let $\omega$ be the assembly of natural numbers equipped with the usual partial order. Then $\mathscr{B} = \left[\omega^\circ, \mathbf{Asm}_{\mathbb{A}}\right]$ is a model of guarded recursion; internal to $\mathscr{B}$ we have a universe $\mathbb{U}$ of presheaves of modest sets, which we may assume to come equipped with a strict extension structure. We may construct a preorder of worlds $\mathbb{W}$ as in the previous sections using $\mathbb{U}$, and we likewise may define the discrete category of heaps $\mathbb{H}$. We define $\mathscr{C}$ to be the full internal subcategory spanned by $\mathbb{U}$; the comprehension $\{-\}$ is simply the inclusion of presheaves of modest sets into presheaves of assemblies. We define $\mathscr{D}$ to be category of algebras for the endofunctor $\blacktriangleright : \mathscr{C} \longrightarrow \mathscr{C}$. The free-forgetful adjunction $\mathsf{U} \dashv \mathsf{F}$ induced by the endofunctor $\blacktriangleright$ then serves as an appropriate input for the polymorphic store construction.

*Remark 6.14.* From Example 6.13 we obtain a model of higher-order store in a topos, and it is natural to compare it to our explicit dependently typed store model from Section 5.4. This is not immediately possible because (1) the construction in this section yields a fibered monad whereas that of Section 5.4 yields an internal monad on a universe, and (2) the model from Section 5.4 takes place in *spans* to account for relational reasoning up to weak bisimulation, whereas we have not accounted for relational reasoning in this section. We may nonetheless compare these two models by (1) restricting the fibered monad to an internal monad, and (2) considering either "hand" of the span model; on this footing, the two models of higher-order store coincide.

## 7 CONCLUSIONS AND FUTURE WORK

*Dependent type theory with higher-order store.* We have defined denotational semantics for higher-order store and polymorphism in impredicative guarded dependent type theory (**iGDTT**); this denotational semantics, in turn, justifies the extension of impredicative guarded dependent type theory with general reference types and a (higher-order) state monad, a theory that we call $\mathbf{iGDTT}^{\mathsf{ref}}$. Although denotational semantics for higher-order state are *a priori* desirable in general, they are essentially mandatory in the case of dependent type theory — an area in which operational methods have proved disappointingly unscalable.

*Relational reasoning atop an intensional equational theory.* The equational theory of store that we model is, however, quite intensional: reads from the store leave behind "abstract steps" that cannot be ignored. Thus in order to support reasoning about stateful computation, we have extended $\mathbf{iGDTT}^{\mathsf{ref}}$ with constructs for proof-relevant relational reasoning based on the ***logical relations as types*** (LRAT) principle: in $\mathbf{iGDTT}^{\mathsf{ref}}_{\mathrm{LRAT}}$, it is possible to exhibit correspondences between stateful computations that combine weak bisimulation for computation steps with parametricity for abstract data types. Because we have "full-spectrum" dependent type theory at our disposal, bisimulation arguments that in prior work required unfolding a very complex operational or denotational semantics can be carried out totally naïvely within the type theory.

### 7.1 Future work

*7.1.1 Swapping and dropping allocations.* Our account of correspondences on worlds and stateful computations is somewhat provisional; although we support storing relational invariants in the heap and working "up to" computation steps, it is not possible in our model to exhibit a correspondence between two programs that *allocate* differently, even if this does not affect their observable behavior. Two things are currently missing from our model:

(1) *Swapping of independent allocations.* To identify $(r \leftarrow \mathsf{new}_{\mathbb{Z}}\,5; s \leftarrow \mathsf{new}_{\mathbb{Z}}\,6; A u\, r\, s)$ with the reordered program $(s \leftarrow \mathsf{new}_{\mathbb{Z}}\,6; r \leftarrow \mathsf{new}_{\mathbb{Z}}\,5; u\, r\, s)$, we would need the relational interpretation of worlds to allow the left-hand world to differ from the right-hand world by a permutation.

(2) *Dropping of inactive allocations.* Ours is a model of *global* state and allocation: thus the denotations of $(r \leftarrow \mathsf{new}_{\mathbb{Z}}\,5; \mathsf{ret}\,())$ and $\mathsf{ret}\,()$ are distinct. One of our goals for future work is to extend our relational layer to account for such identifications, potentially by adapting the ideas of Kammar et al. [2017] who have themselves extended the classic Plotkin–Power account of local state [Plotkin and Power 2002] to support storage of pointers. We believe such an adaptation is plausible, but many questions remain unanswered in the space between storage of ground types and pointers with syntactic worlds and full thunk storage with semantic worlds.

We believe that it may be possible to address the limitations outlined above using *nominal* techniques [Gabbay and Pitts 2002; Pitts 2016, 2013]; it is also worth considering the relationship between the *logical relations as types* principle and the proof-relevant logical relations for swapping and dropping employed by Benton et al. [2013].

*7.1.2 More sophisticated Kripke worlds.* We also expect that it may be profitable to build on prior work in the operational semantics of state [Ahmed et al. 2009; Dreyer et al. 2010] involving yet more sophisticated Kripke worlds to facilitate local reasoning on the heap. Such sophistication inevitably draws one in the direction of proper program logics, however, which is another area of future work.

*7.1.3 Higher-order separation logic for modular reasoning about stateful programs.* Closely related to the question of Kripke worlds broached above, it is reasonable to consider layering atop our type theory the higher-order separation logic of Iris [Jung et al. 2015] to support modular reasoning about stateful programs. Semantically there is no obstacle to defining the Kripke worlds in terms of a resource algebra that incorporates higher-order ghost state [Bizjak and Birkedal 2018], but it remains an open question how to surface the details of this Kripke model in dependent type theory in a useful way. Although logic-enriched dependent type theory is well-understood [Jacobs 1999; Phoa 1992; Taylor 1999], we expect that satisfactorily developing the theory of higher-order separation logic over dependent types may require some new ideas.

## ACKNOWLEDGMENTS

We are thankful to Robert Harper and Rasmus Møgelberg for helpful conversations concerning this project. This work was supported in part by a Villum Investigator grant (no. 25804), Center for Basic Research in Program Verification (CPV), from the VILLUM Foundation. Jonathan Sterling is funded by the European Union under the Marie Skłodowska-Curie Actions Postdoctoral Fellowship project *TypeSynth: synthetic methods in program verification*. Views and opinions expressed are however those of the authors only and do not necessarily reflect those of the European Union or the European Commission. Neither the European Union nor the granting authority can be held responsible for them.

## REFERENCES

Martín Abadi and Gordon D. Plotkin. 2019. A Simple Differentiable Programming Language. *Proceedings of the ACM on Programming Languages* 4, POPL (Dec. 2019). https://doi.org/10.1145/3371106

Michael Abbott, Thorsten Altenkirch, and Neil Ghani. 2005. Containers: Constructing strictly positive types. *Theoretical Computer Science* 342, 1 (2005), 3–27. https://doi.org/10.1016/j.tcs.2005.06.002 Applied Semantics: Selected Topics.

Amal Ahmed, Derek Dreyer, and Andreas Rossberg. 2009. State-dependent representation independence. In *Proceedings of the 36th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2009, Savannah, GA, USA, January 21-23, 2009*, Zhong Shao and Benjamin C. Pierce (Eds.). Association for Computing Machinery, 340–353. https://doi.org/10.1145/1480881.1480925

Amal Jamil Ahmed. 2004. *Semantics of Types for Mutable State.* Ph. D. Dissertation. Princeton University. http://www.cs.indiana.edu/~amal/ahmedsthesis.pdf

Pierre America and Jan J. M. M. Rutten. 1987. Solving Reflexive Domain Equations in a Category of Complete Metric Spaces. In *Proceedings of the 3rd Workshop on Mathematical Foundations of Programming Language Semantics.* Springer-Verlag, Berlin, Heidelberg, 254–288.

Andrew W. Appel, Paul-André Melliès, Christopher D. Richards, and Jérôme Vouillon. 2007. A Very Modal Model of a Modern, Major, General Type System. In *Proceedings of the 34th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages.* Association for Computing Machinery, Nice, France, 109–122.

Steve Awodey. 2021. A Quillen model structure on the category of cartesian cubical sets. (2021). https://github.com/awodey/math/blob/e8c715cc5cb6a966e736656bbe54d0483f9650fc/QMS/qms.pdf Unpublished notes.

Steve Awodey, Jonas Frey, and Sam Speight. 2018. Impredicative Encodings of (Higher) Inductive Types. In *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science.* Association for Computing Machinery, Oxford, United Kingdom, 76–85. https://doi.org/10.1145/3209108.3209130

Nick Benton, Martin Hofmann, and Vivek Nigam. 2013. Proof-Relevant Logical Relations for Name Generation. In *Typed Lambda Calculi and Applications*, Masahito Hasegawa (Ed.). Springer Berlin Heidelberg, Berlin, Heidelberg, 48–60.

Jean-Philippe Bernardy, Patrik Jansson, and Ross Paterson. 2012. Proofs for Free: Parametricity for Dependent Types. *J. Funct. Program.* 22, 2 (March 2012), 107–152. https://doi.org/10.1017/S0956796812000056

Jean-Philippe Bernardy and Guilhem Moulin. 2012. A Computational Interpretation of Parametricity. In *Proceedings of the 2012 27th Annual IEEE/ACM Symposium on Logic in Computer Science.* IEEE Computer Society, New Orleans, Louisiana, 135–144. https://doi.org/10.1109/LICS.2012.25

L. Birkedal and R. E. Møgelberg. 2013. Intensional Type Theory with Guarded Recursive Types qua Fixed Points on Universes. In *Proceedings of the 2013 28th Annual ACM/IEEE Symposium on Logic in Computer Science.* IEEE Computer Society, Washington, DC, USA, 213–222. https://doi.org/10.1109/LICS.2013.27

Lars Birkedal, Rasmus Ejlers Møgelberg, Jan Schwinghammer, and Kristian Støvring. 2011a. First Steps in Synthetic Guarded Domain Theory: Step-Indexing in the Topos of Trees. In *Proceedings of the 2011 IEEE 26th Annual Symposium on Logic in Computer Science*. IEEE Computer Society, Washington, DC, USA, 55–64. https://doi.org/10.1109/LICS.2011.16 arXiv:1208.3596 [cs.LO]

Lars Birkedal, Bernhard Reus, Jan Schwinghammer, Kristian Støvring, Jacob Thamsborg, and Hongseok Yang. 2011b. Step-Indexed Kripke Models over Recursive Worlds. In *Proceedings of the 38th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*. Association for Computing Machinery, Austin, Texas, USA, 119–132. https://doi.org/10.1145/1926385.1926401

Lars Birkedal, Kristian Støvring, and Jacob Thamsborg. 2010. Realisability semantics of parametric polymorphism, general references and recursive types. *Mathematical Structures in Computer Science* 20, 4 (2010), 655–703. https://doi.org/10.1017/S0960129510000162

Aleš Bizjak and Lars Birkedal. 2018. On Models of Higher-Order Separation Logic. *Electronic Notes in Theoretical Computer Science* 336 (2018), 57–78. https://doi.org/10.1016/j.entcs.2018.03.016

Aleš Bizjak, Hans Bugge Grathwohl, Ranald Clouston, Rasmus E. Møgelberg, and Lars Birkedal. 2016. Guarded Dependent Type Theory with Coinductive Types. In *Foundations of Software Science and Computation Structures: 19th International Conference, FOSSACS 2016, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2016, Eindhoven, The Netherlands, April 2–8, 2016, Proceedings*, Bart Jacobs and Christof Löding (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 20–35. https://doi.org/10.1007/978-3-662-49630-5_2 arXiv:1601.01586 [cs.LO]

Francis Borceux. 1994. *Handbook of Categorical Algebra 1 – Basic Category Theory*. Encyclopedia of Mathematics and its Applications, Vol. 1. Cambridge University Press. https://doi.org/10.1017/CBO9780511525858

Edwin Brady. 2021. Idris 2: Quantitative Type Theory in Practice. In *35th European Conference on Object-Oriented Programming (ECOOP 2021) (Leibniz International Proceedings in Informatics (LIPIcs), Vol. 194)*, Anders Møller and Manu Sridharan (Eds.). Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl, Germany, 9:1–9:26. https://doi.org/10.4230/LIPIcs.ECOOP.2021.9

The Coq Development Team. 2016. The Coq Proof Assistant Reference Manual.

Thierry Coquand. 1986. An Analysis of Girard's Paradox. In *Proceedings of the First Symposium on Logic in Computer Science*. IEEE Computer Society, 227–236.

Thierry Coquand, Carl Gunter, and Glynn Winskel. 1994. Domain Theoretic Models Of Polymorphism. *Information and Computation* 81 (June 1994). https://doi.org/10.1016/0890-5401(89)90068-0

Leonardo De Moura and Sebastian Ullrich. 2021. The Lean 4 Theorem Prover and Programming Language (System Description). (2021). To appear in the proceedings of the 28th International Conference on Automated Deduction.

Derek Dreyer, Georg Neis, Andreas Rossberg, and Lars Birkedal. 2010. A Relational Modal Logic for Higher-Order Stateful ADTs. In *Proceedings of the 37th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL '10)*. Association for Computing Machinery, Madrid, Spain, 185–198. https://doi.org/10.1145/1706299.1706323

Martín Escardó. 1999. A metric model of PCF. In *Workshop on Realizability Semantics and Applications*.

Martín Hötzel Escardó. 2021. Injective types in univalent mathematics. *Mathematical Structures in Computer Science* 31, 1 (2021), 89–111. https://doi.org/10.1017/S0960129520000225

Murdoch J. Gabbay and Andrew M. Pitts. 2002. A New Approach to Abstract Syntax with Variable Binding. *Formal Aspects of Computing* 13, 3 (July 2002), 341–363. https://doi.org/10.1007/s001650200016

Daniel Gratzer, Michael Shulman, and Jonathan Sterling. 2022. Strict universes for Grothendieck topoi. (Feb. 2022). https://doi.org/10.48550/arXiv.2202.12012 arXiv:2202.12012 [math.CT] Unpublished manuscript.

Martin Hofmann. 1997. Syntax and Semantics of Dependent Types. In *Semantics and Logics of Computation*, Andrew M. Pitts and Peter Dybjer (Eds.). Cambridge University Press, 79–130.

Martin Hofmann and Thomas Streicher. 1997. Lifting Grothendieck Universes. (1997). https://www2.mathematik.tu-darmstadt.de/~streicher/NOTES/lift.pdf Unpublished note.

J. M. E. Hyland. 1982. The effective topos. In *The L.E.J. Brouwer Centenary Symposium*, A. S. Troelstra and D. Van Dalen (Eds.). North Holland Publishing Company, 165–216.

J. M. E. Hyland. 1988. A small complete category. *Annals of Pure and Applied Logic* 40, 2 (1988), 135–165. https://doi.org/10.1016/0168-0072(88)90018-8

J. M. E. Hyland, E. P. Robinson, and G. Rosolini. 1990. The Discrete Objects in the Effective Topos. *Proceedings of the London Mathematical Society* s3-60, 1 (Jan. 1990), 1–36. https://doi.org/10.1112/plms/s3-60.1.1

Bart Jacobs. 1999. *Categorical Logic and Type Theory*. Number 141 in Studies in Logic and the Foundations of Mathematics. North Holland, Amsterdam.

Peter T. Johnstone, Stephen Lack, and Paweł Sobociński. 2007. Quasitoposes, Quasiadhesive Categories and Artin Glueing. In *Algebra and Coalgebra in Computer Science*, Till Mossakowski, Ugo Montanari, and Magne Haveraaen (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 312–326.

Simon Peyton Jones. 2001. *Tackling the awkward squad: monadic input/output, concurrency, exceptions, and foreign-language calls in Haskell.* IOS Press, 47–96.

Ralf Jung, Robbert Krebbers, Jacques-Henri Jourdan, Aleš Bizjak, Lars Birkedal, and Derek Dreyer. 2018. Iris from the ground up: A modular foundation for higher-order concurrent separation logic. *Journal of Functional Programming* 28 (2018), e20. https://doi.org/10.1017/S0956796818000151

Ralf Jung, David Swasey, Filip Sieczkowski, Kasper Svendsen, Aaron Turon, Lars Birkedal, and Derek Dreyer. 2015. Iris: Monoids and Invariants As an Orthogonal Basis for Concurrent Reasoning. In *POPL '15: Proceedings of the 42nd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages.* Association for Computing Machinery, Mumbai, India, 637–650. https://doi.org/10.1145/2676726.2676980

Ohad Kammar, Paul Blain Levy, Sean K. Moss, and Sam Staton. 2017. A monad for full ground reference cells. (2017). arXiv:1702.04908 [cs.PL]

Neelakantan R. Krishnaswami, Pierre Pradic, and Nick Benton. 2015. Integrating Linear and Dependent Types. In *POPL '15: Proceedings of the 42nd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages.* Association for Computing Machinery, Mumbai, India, 17–30.

Paul Levy. 2002. Possible World Semantics for General Storage in Call-By-Value. 232–246. https://doi.org/10.1007/3-540-45793-3_16

Paul Levy. 2003a. *Call-by-Push-Value: A Functional/Imperative Synthesis.* Kluwer, Semantic Structures in Computation, 2.

Paul Blain Levy. 2003b. Adjunction Models For Call-By-Push-Value With Stacks. *Electronic Notes in Theoretical Computer Science* 69 (2003), 248–271. https://doi.org/10.1016/S1571-0661(04)80568-1 CTCS'02, Category Theory and Computer Science.

Paul Blain Levy. 2004. *Call-By-Push-Value: A Functional/Imperative Synthesis (Semantics Structures in Computation, V. 2).* Kluwer Academic Publishers, Norwell, MA, USA.

Peter Lietz and Thomas Streicher. 2002. Impredicativity Entails Untypedness. *Mathematical Structures in Computer Science* 12, 3 (June 2002), 335–347. https://doi.org/10.1017/S0960129502003663

Conor McBride and Ross Paterson. 2008. Applicative Programming with Effects. *Journal of Functional Programming* 18, 1 (Jan. 2008), 1–13. https://doi.org/10.1017/S0956796807006326

Rasmus Ejlers Møgelberg and Marco Paviotti. 2016. Denotational Semantics of Recursive Types in Synthetic Guarded Domain Theory. In *Proceedings of the 31st Annual ACM/IEEE Symposium on Logic in Computer Science.* Association for Computing Machinery, New York, NY, USA, 317–326. https://doi.org/10.1145/2933575.2934516

Eugenio Moggi. 1991. Notions of computation and monads. *Information and Computation* 93, 1 (1991), 55–92. https://doi.org/10.1016/0890-5401(91)90052-4 Selections from 1989 IEEE Symposium on Logic in Computer Science.

Ulf Norell. 2009. Dependently Typed Programming in Agda. In *Proceedings of the 4th International Workshop on Types in Language Design and Implementation (TLDI '09).* Association for Computing Machinery, Savannah, GA, USA, 1–2.

Frank J. Oles. 1986. *Type Algebras, Functor Categories and Block Structure.* Cambridge University Press, USA, 543–573.

Ian Orton and Andrew M. Pitts. 2016. Axioms for Modelling Cubical Type Theory in a Topos. In *25th EACSL Annual Conference on Computer Science Logic (CSL 2016) (Leibniz International Proceedings in Informatics (LIPIcs), Vol. 62)*, Jean-Marc Talbot and Laurent Regnier (Eds.). Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany, 24:1–24:19. https://doi.org/10.4230/LIPIcs.CSL.2016.24

Daniele Palombi and Jonathan Sterling. 2022. Classifying topoi in synthetic guarded domain theory. In *Proceedings 38th Conference on Mathematical Foundations of Programming Semantics, MFPS 2022.* https://www.jonmsterling.com/papers/palombi-sterling:2022.pdf To appear.

Marco Paviotti. 2016. *Denotational semantics in Synthetic Guarded Domain Theory.* Ph. D. Dissertation. IT-Universitetet i København, Denmark.

Marco Paviotti, Rasmus Ejlers Møgelberg, and Lars Birkedal. 2015. A Model of PCF in Guarded Type Theory. *Electronic Notes in Theoretical Computer Science* 319, Supplement C (2015), 333–349. https://doi.org/10.1016/j.entcs.2015.12.020 The 31st Conference on the Mathematical Foundations of Programming Semantics (MFPS XXXI).

Pierre-Marie Pédrot and Nicolas Tabareau. 2019. The Fire Triangle: How to Mix Substitution, Dependent Elimination, and Effects. *Proceedings of the ACM on Programming Languages* 4, POPL (Dec. 2019). https://doi.org/10.1145/3371126

W. K.-S. Phoa. 1992. *An Introduction to Fibrations, Topos Theory, the Effective Topos and Modest Sets.* Technical Report ECS-LFCS-92-208. Department of Computer Science, University of Edinburgh.

Andrew Pitts. 2016. Nominal Techniques. *ACM SIGLOG News* 3, 1 (Feb. 2016), 57–72. https://doi.org/10.1145/2893582.2893594

Andrew M. Pitts. 2013. *Nominal Sets: Names and Symmetry in Computer Science.* Cambridge University Press, New York, NY, USA.

Gordon D. Plotkin and John Power. 2002. Notions of Computation Determine Monads. In *Proceedings of the 5th International Conference on Foundations of Software Science and Computation Structures.* Springer-Verlag, Berlin, Heidelberg, 342–356.

John Power. 2011. *Indexed Lawvere theories for local state.* American Mathematical Society, USA United States, 213–229.

John C. Reynolds. 1981. *The Essence of Algol.* North-Holland, Amsterdam.

Egbert Rijke, Michael Shulman, and Bas Spitters. 2020. Modalities in homotopy type theory. *Logical Methods in Computer Science* Volume 16, Issue 1 (Jan. 2020). https://doi.org/10.23638/LMCS-16(1:2)2020 arXiv:1706.07526 [math.CT]

Jan J. M. M. Rutten and Daniele Turi. 1992. On the Foundation of Final Semantics: Non-Standard Sets, Metric Spaces, Partial Orders. In *Proceedings of the REX Workshop on Semantics: Foundations and Applications*. Springer-Verlag, Berlin, Heidelberg, 477–530.

Jonathan Sterling. 2021. *First Steps in Synthetic Tait Computability: The Objective Metatheory of Cubical Type Theory.* Ph. D. Dissertation. Carnegie Mellon University. https://doi.org/10.5281/zenodo.6990769 Version 1.1, revised May 2022.

Jonathan Sterling and Carlo Angiuli. 2021. Normalization for Cubical Type Theory. In *2021 36th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*. IEEE Computer Society, Los Alamitos, CA, USA, 1–15. https://doi.org/10.1109/LICS52264.2021.9470719 arXiv:2101.11479 [cs.LO]

Jonathan Sterling and Robert Harper. 2021. Logical Relations as Types: Proof-Relevant Parametricity for Program Modules. *J. ACM* 68, 6 (Oct. 2021). https://doi.org/10.1145/3474834 arXiv:2010.08599 [cs.PL]

Jonathan Sterling and Robert Harper. 2022. Sheaf semantics of termination-insensitive noninterference. In *7th International Conference on Formal Structures for Computation and Deduction (FSCD 2022) (Leibniz International Proceedings in Informatics (LIPIcs), Vol. 228)*, Amy P. Felty (Ed.). Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany, 5:1–5:19. https://doi.org/10.4230/LIPIcs.FSCD.2022.5 arXiv:2204.09421 [cs.PL]

Thomas Streicher. 2017. Realizability. (2017). https://www2.mathematik.tu-darmstadt.de/~streicher/REAL/REAL.pdf Lecture notes.

Kasper Svendsen and Lars Birkedal. 2014. Impredicative Concurrent Abstract Predicates. In *Programming Languages and Systems*, Zhong Shao (Ed.). Springer Berlin Heidelberg, Berlin, Heidelberg, 149–168.

Kasper Svendsen, Lars Birkedal, and Aleksandar Nanevski. 2011. Partiality, State and Dependent Types. In *Typed Lambda Calculi and Applications - 10th International Conference, TLCA 2011, Novi Sad, Serbia, June 1-3, 2011. Proceedings (Lecture Notes in Computer Science, Vol. 6690)*, C.-H. Luke Ong (Ed.). Springer, 198–212. https://doi.org/10.1007/978-3-642-21691-6_17

Paul Taylor. 1996. Intuitionistic sets and ordinals. *The Journal of Symbolic Logic* 61, 3 (1996), 705–744. https://doi.org/10.2307/2275781

Paul Taylor. 1999. *Practical Foundations of Mathematics.* Cambridge University Press, Cambridge, New York (N. Y.), Melbourne.

Jacob Junker Thamsborg. 2010. *Denotational World-indexed Logical Relations and Friends.* Ph. D. Dissertation. IT-Universitetet i København, Denmark.

Matthijs Vákár. 2017. *In Search of Effectful Dependent Types.* Ph. D. Dissertation. University of Oxford. https://doi.org/10.48550/arXiv.1706.07997 arXiv:1706.07997 [cs.LO]

Matthijs Vákár, Ohad Kammar, and Sam Staton. 2019. A Domain Theory for Statistical Probabilistic Programming. *Proceedings of the ACM on Programming Languages* 3, POPL (Jan. 2019). https://doi.org/10.1145/3290349

Jaap van Oosten. 2008. *Realizability: An Introduction to its Categorical Side.* Elsevier Science, San Diego.

Li-yao Xia, Yannick Zakowski, Paul He, Chung-Kil Hur, Gregory Malecha, Benjamin C. Pierce, and Steve Zdancewic. 2019. Interaction Trees: Representing Recursive and Impure Programs in Coq. *Proceedings of the ACM on Programming Languages* 4, POPL (Dec. 2019). https://doi.org/10.1145/3371119

# A SYNTAX OF POLYMORPHIC CALL-BY-PUSH-VALUE

In this section we describe the syntax of a version of call-by-push-value with polymorphism and higher-order store. First we describe the structure of contexts and judgments:

(1) $\Psi \vdash$ means that $\Psi$ is a type context.
(2) $\Psi \vdash A\ \mathit{vtype}$ means that $A$ is a value type in context $\Psi$.
(3) $\Psi \vdash X\ \mathit{ctype}$ means that $X$ is a computation type in context $\Psi$.
(4) $\Psi; \Gamma \vdash$ means that $\Gamma$ is a value context in type context $\Psi$.
(5) $\Psi; \Gamma \vdash V : A$ means that $V$ is a value of type $A$ in context $\Psi; \Gamma$.
(6) $\Psi; \Gamma \mid X \vdash K : Y$ means that $K$ is a *stack* from $X$ and $Y$ in context $\Psi; \Gamma$.
(7) $\Psi; \Gamma \vdash M : X$ means that $M$ is computation of type $X$ in context $\Psi; \Gamma$.

The structure of contexts is described below; contexts contain type variables and value variables.

$$\frac{}{\cdot \vdash} \qquad \frac{\Psi \vdash \qquad (\alpha \notin \Psi)}{\Psi, \alpha \vdash} \qquad \frac{\Psi; \Gamma \vdash \qquad \Psi \vdash A\ \mathit{vtype} \qquad (x \notin \Gamma)}{\Psi; \Gamma, x : A \vdash} \qquad \frac{\alpha \in \Psi}{\Psi \vdash \alpha\ \mathit{vtype}} \qquad \frac{(x : A) \in \Gamma}{\Psi; \Gamma \vdash x : A}$$

## A.1 Stack composition

$$\frac{}{\Psi; \Gamma \mid X \vdash \blacksquare : X} \qquad \frac{\Psi; \Gamma \mid X \vdash K : Y \qquad \Psi; \Gamma \mid Y \vdash L : Z}{\Psi; \Gamma \mid X \vdash K; L : Z}$$

$$\frac{\Psi; \Gamma \mid W \vdash J : X \qquad \Psi; \Gamma \mid X \vdash K : Y \qquad \Psi; \Gamma \mid Y \vdash L : Z}{\Psi; \Gamma \mid W \vdash J; (K; L) \equiv (J; K); L : Z} \qquad \frac{\Psi; \Gamma \mid X \vdash KY}{\Psi; \Gamma \mid X \vdash \blacksquare; K \equiv KY}$$

$$\frac{\Psi; \Gamma \mid X \vdash KY}{\Psi; \Gamma \mid X \vdash K; \blacksquare \equiv KY}$$

## A.2 Stack dismantling

$$\frac{\Psi; \Gamma \vdash M : X \qquad \Psi; \Gamma \mid X \vdash K : Y}{\Psi; \Gamma \vdash M \diamond K : Y} \qquad \frac{\Psi; \Gamma \vdash M : X}{\Psi; \Gamma \vdash M \diamond \blacksquare \equiv M : X}$$

$$\frac{\Psi; \Gamma \vdash M : X \qquad \Psi; \Gamma \mid X \vdash K : Y \qquad \Psi; \Gamma \mid Y \vdash L : Z}{\Psi; \Gamma \vdash M \diamond K; L \equiv (M \diamond K) \diamond L : Z}$$

## A.3 Left adjunctives

$$\frac{\Psi \vdash A\ \mathit{vtype}}{\Psi \vdash \mathsf{F}A\ \mathit{ctype}} \qquad \frac{\Psi; \Gamma \vdash V : A}{\Psi; \Gamma \vdash \mathsf{ret}\, V : \mathsf{F}A} \qquad \frac{\Psi; \Gamma, x : A \vdash N[x] : X}{\Psi; \Gamma \mid \mathsf{F}A \vdash \tilde{\mu}x.\, N[x] : X}$$

$$\frac{\Psi; \Gamma \vdash V : A \qquad \Psi; \Gamma, x : A \vdash N[x] : X}{\Psi; \Gamma \vdash \mathsf{ret}\, V \diamond \tilde{\mu}x.\, N[x] \equiv N[V] : X} \qquad \frac{\Psi; \Gamma \mid \mathsf{F}A \vdash K : X}{\Psi; \Gamma \mid \mathsf{F}A \vdash K \equiv \tilde{\mu}x.\, \mathsf{ret}\, x \diamond K : X}$$

## A.4 Right adjunctives

$$\frac{\Psi \vdash X\ \mathit{ctype}}{\Psi \vdash \mathsf{U}X\ \mathit{vtype}} \qquad \frac{\Psi; \Gamma \vdash M : X}{\Psi; \Gamma \vdash \mathsf{thunk}\, M : \mathsf{U}X} \qquad \frac{\Psi; \Gamma \vdash V : \mathsf{U}X}{\Psi; \Gamma \vdash \mathsf{force}\, V : X}$$

$$\frac{\Psi; \Gamma \vdash M : X}{\Psi; \Gamma \vdash \mathsf{force}\,(\mathsf{thunk}\, M) \equiv M : X} \qquad \frac{\Psi; \Gamma \vdash V : \mathsf{U}X}{\Psi; \Gamma \vdash V \equiv \mathsf{thunk}\,(\mathsf{force}\, V) : \mathsf{U}X}$$

## A.5 Universal types

$$\frac{\Psi, \alpha \vdash X[\alpha]\ \mathit{ctype}}{\Psi \vdash \forall_\alpha X[\alpha]\ \mathit{ctype}} \qquad \frac{\Psi, \alpha; \Gamma \mid X \vdash K[\alpha] : Y[\alpha] \qquad (\alpha \notin \Psi; \Gamma)}{\Psi; \Gamma \mid X \vdash \mathsf{pop}\,\alpha.\, K[\alpha] : \forall_\alpha Y[\alpha]}$$

$$\frac{\Psi \vdash A\ \mathit{vtype} \qquad \Psi \mid X \vdash K : \forall_\alpha Y[\alpha]}{\Psi; \Gamma \mid X \vdash \mathsf{push}\,A; K : Y[A]} \qquad \frac{\Psi \vdash A\ \mathit{vtype} \qquad \Psi, \alpha; \Gamma \mid X \vdash K[\alpha] : Y[\alpha]}{\Psi; \Gamma \mid X \vdash \mathsf{push}\,A; \mathsf{pop}\,\alpha.\, K[\alpha] \equiv K[\alpha] : Y[A]}$$

$$\frac{\Psi; \Gamma \mid X \vdash K : \forall_\alpha Y[\alpha]}{\Psi; \Gamma \mid X \vdash K \equiv \mathsf{pop}\,\alpha.\, \mathsf{push}\,\alpha; K : \forall_\alpha Y[\alpha]}$$

## A.6 Reference types and computational effects

$$\frac{\Psi \vdash A\ \mathit{vtype}}{\Psi \vdash \mathsf{ref}\,A\ \mathit{vtype}} \qquad \frac{\Psi; \Gamma \vdash M : A}{\Psi; \Gamma \vdash \mathsf{step}; M : X} \qquad \frac{\Psi; \Gamma \vdash V : A \qquad \Psi; \Gamma, x : \mathsf{ref}\,A \vdash M[x] : X}{\Psi; \Gamma \vdash \mathsf{new}\,x := V\ \mathsf{in}\ M[x] : X}$$

$$\frac{\Psi; \Gamma \vdash V : \mathsf{ref}\,A \qquad \Psi; \Gamma, x : A \vdash M[x] : X}{\Psi; \Gamma \vdash x\ ?\ V.\, M[x]} \qquad \frac{\Psi; \Gamma \vdash V : \mathsf{ref}\,A \qquad \Psi; \Gamma \vdash W : A \qquad \Psi; \Gamma \vdash M : X}{\Psi; \Gamma \vdash V := W; M : X}$$

$$\frac{\Psi; \Gamma \vdash V : A \qquad \Psi; \Gamma \vdash W : A \qquad \Psi; \Gamma, x : \mathsf{ref}\,A \vdash M[x] : X}{\Psi; \Gamma \vdash (\mathsf{new}\,x := V\ \mathsf{in}\ x := W; M[x]) \equiv (\mathsf{new}\,x := W\ \mathsf{in}\ M[x]) : X}$$

$$\frac{\Psi; \Gamma \vdash V : \mathsf{ref}\,A \qquad \Psi; \Gamma \vdash W : A \qquad \Psi; \Gamma, x : A \vdash M[x] : X}{\Psi; \Gamma \vdash (V := W; x\ ?\ V.\, M[x]) \equiv (V := W; \mathsf{step}; M[W]) : X}$$

$$\frac{\Psi; \Gamma \vdash U : \mathsf{ref}\,A \qquad \Psi; \Gamma \vdash V, W : A \qquad \Psi; \Gamma \vdash MX}{\Psi; \Gamma \vdash (U := V; U := W; M) \equiv (U := W; M) : X}$$

We omit the rules that make step commute with all the effects described above.

## A.7 The unit type

To pass between computations and stacks, we need to include a unit type.

$$\frac{}{\Gamma \vdash 1\ \mathit{vtype}} \qquad \frac{}{\Gamma \vdash () : 1} \qquad \frac{\Gamma \vdash V : 1}{\Gamma \vdash V \equiv () : 1}$$

## A.8 Macro definitions

We include a few "macro expansions" to support more familiar notations for the focalized constructs of the core calculus.

$$\begin{aligned}
(x \leftarrow M; N[x]) &\triangleq M \diamond \tilde{\mu} x.\, N[x] \\
(\Lambda\alpha. M[\alpha]) &\triangleq \mathsf{ret}\ () \diamond \mathsf{pop}\,\alpha.\, \tilde{\mu}\_.\, M[\alpha] \\
M(A) &\triangleq \mathsf{push}\,A; \mathsf{ret}\ () \diamond \mathsf{push}\,A; \tilde{\mu}\_.\, M[\alpha]
\end{aligned}$$