Free theorems from univalent reference types

The impact of univalence on denotational semantics

Jonathan Sterling
Aarhus University

Daniel Gratzer
Aarhus University

Lars Birkedal
Aarhus University

Abstract

We develop a denotational semantics for general reference types in an impredicative version of ***guarded homotopy type theory***, an adaptation of synthetic guarded domain theory to Voevodsky's univalent foundations. We observe for the first time the profound impact of univalence on the denotational semantics of mutable state. Univalence automatically ensures that all computations are invariant under symmetries of the heap—a bountiful source of free theorems. In particular, even the most simplistic univalent model enjoys many new program equivalences that do not hold when the same constructions are carried out in the universes of traditional set-level (extensional) type theory.

2012 ACM Subject Classification Theory of computation → Denotational semantics; Theory of computation → Categorical semantics; Theory of computation → Type structures; Theory of computation → Type theory

Keywords and phrases univalent foundations, homotopy type theory, impredicative encodings, synthetic guarded domain theory, guarded recursion, higher-order store, reference types

Funding 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: 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.

Acknowledgements We are thankful to Carlo Angiuli, Steve Awodey, and Robert Harper for teaching us the importance of realizability methods in homotopy type theory. We thank Zhixuan Yang for proof-reading.

1 Introduction

Moggi [32] famously distinguished three semantics-based approaches to proving equivalences between programs: ***operational***, ***denotational***, and ***logical***. Operational semantics studies programs *indirectly* by investigating the properties of a transition function that executes programs *qua* code on a highly specific idealized computer; in contrast, denotational semantics views programs *directly* as functions on highly specialized kinds of spaces, without making any detour through transition functions. Moggi's departure is to advance a *logical* approach to program equivalence, in which a programming language is an equational theory equipped with a *category* of denotational models for which it is both sound and complete.

Moggi's logical approach to program equivalence therefore subsumes traditional denotational semantics: both the general and the particular necessarily exude their own depth and sophistication, but they are now correctly situated in relation to each other so that workers in semantics can reap the greatest benefits from the *theory–model* dialectic:

1. Even if there is a distinguished "standard" model of a given programming language (*e.g.* the Scott model of PCF), any non-trivial investigation of the syntax of that language necessarily involves non-standard models — if only because induction can always be seen as a model construction. These non-standard models include the *generic* model, built from the theory itself, as well as models based on logical relations; thus the need for clear thinking about many models via logical semantics cannot be bypassed.
2. Conversely, the discovery of a new model of a programming language can inspire and justify the refinement of its equational theory: for instance, *parametric models* have been used to justify equational theories for data abstraction and local store. In the other direction, the discovery of a "non-model" that nonetheless has desirable properties can open up new semantic vistas by motivating a relaxed equational theory.

1.1 State and reference types: static and dynamic allocation

One of the oldest programming constructs is *state*: the ability to read from and write to the computer's memory as a side effect. Theories of state delineate themselves along two axes: (1) the kinds of data that can be stored, and (2) the kinds of allocations allowed. On the first axis, languages range from being able to store integers and strings (***first-order store***) all the way to being able to store elements of arbitrary types, including closures (***higher-order store***). On the second axis, we have ***static allocation*** on one end, where the type of a function specifies exactly what kind of state it uses, and ***dynamic allocation*** on the other end, where the types and quantity of memory cells allocated are revealed only during execution. Under dynamic allocation, one has ***reference types*** whose elements are *pointers* to memory cells storing elements of a given type.

1.2 Equational theories of dynamic storage: between local and global

The semantics of state are only difficult under dynamic allocation; indeed, computations that interact with a statically known heap configuration $\overline{\ell_i : \sigma_i}$ can be classified by Moggi's state monad $\sigma \to \sigma \times -$ where $\sigma :\equiv \prod_i \sigma_i$, and it is reasonable to *define* the equational theory of static allocation by means of this interpretation. The equational theory of *dynamic* storage is by contrast far from solidified: the introduction of dynamic allocation opens up a spectrum of abstraction between what may be called ***local store*** and ***global store***.

Global store is the least abstract theory of dynamic allocation: in a model of global store, it is permitted that allocations be globally observable regardless of their impact on the results of computations. For instance, global store models are allowed to distinguish the program $(\ell \leftarrow \mathsf{alloc}\,\texttt{"hello"}; \mathsf{ret}\,10)$ from the simpler program $\mathsf{ret}\,10$. By contrast, models of *local store* validate equations resembling an idealized garbage collector, in which the heap is only observable through its abstract interface read/write interface; in a model of local store, we necessarily have $(\ell \leftarrow \mathsf{alloc}\,\texttt{"hello"}; \mathsf{ret}\,10) = \mathsf{ret}\,10$ as well as many other equations.

The abstraction offered by local store is highly desirable. Moreover, Staton [42] has shown that Plotkin and Power's algebraic theory of *first-order* local store [38] is complete in the extremely strong sense that it derives any consistent equation. Beyond first-order references, the very definition of the local store theory becomes less clear and a variety of theories have emerged that are inspired by what equations we can find actual models of. For example, Kammar *et al.* [27] have constructed a compelling model of local ***full ground store***, going beyond first-order store by allowing pointers to pointers. On the other hand, Levy [28, 29, 30] has given a domain theoretic model of the *global* allocation theory of higher-order store.

1.3 Semantic worlds and guarded models of higher-order store

Denotational models of full dynamic allocation, such as those of Plotkin and Power [38], Levy [28], and Kammar *et al.* [27], tend to share an important defect: in the model, a semantic program can only allocate a memory cell with a *syntactic* type. This restriction is quite unnatural and impractical in the context of higher-order store, where many important program equivalences actually follow from the presence of exotic semantic types lying outside the image of the interpretation function (*e.g.* in relational models à la Girard and Reynolds).

The search for models of general references closed under allocation of cells with *semantic* types has been major motivation of current work in *guarded domain theory*, expressed in operational semantics by *step-indexing* [8, 3] and in denotational semantics by means of various generalizations of metric space [10, 6, 21, 17]. The problem solved by guarded domain theory is the following famous circularity described in several prior works [3, 9, 17]:

1. A semantic type needs to be some kind of covariant family of predomains indexed in the possible configurations of the heap ("worlds"); a single predomain won't do, because the elements of type $\mathsf{IORef}\,\sigma$ vary depending on what cells have been allocated.
2. A semantic world should be a finite mapping from memory locations to semantic types.

Guarded domain theory approximates a solution to the domain equation evoked above by decreasing precision at every recursive occurrence. Although it may be possible to find a fully precise solution to this domain equation using traditional domain theory, Birkedal *et al.* [18, §5] have shown that such a solution will *not* brook the interpretation of reference types by a continuous function on the domain of all types, ruling out semantics for recursive types. Thus guarded domain theory or step-indexing would seem to be mandatory for *functional* models of general reference types with semantic worlds.[1]

Models of guarded domain theory can be embedded into topoi whose internal language is referred to as ***synthetic guarded domain theory*** or *SGDT*; the most famous of these topoi is the ***topos of trees*** [17] given by presheaves on ω. The idea of using synthetic guarded domain theory as a setting for the naïve denotational semantics of programming languages with general recursion was first explored by Paviotti, Møgelberg, and Birkedal [36, 31, 35].

Sterling, Gratzer, and Birkedal [43] have recently extended the program of Paviotti *et al.* to the general case of full higher-order store with polymorphism and recursive types: in particular, *op. cit.* have shown how to model general reference types in synthetic guarded domain theory assuming an impredicative universe (as can be found in realizability models [25, 26]). The model of *op. cit.* is the starting point of the present paper: by adapting the construction of Sterling *et al.* to the setting of univalent foundations, we obtain a new suite of equational reasoning principles that we refer to as the *theory of univalent reference types*.

1.4 Univalent reference types and free theorems in the heap

The thesis of this paper is that Voevodsky's *univalence* principle leads to simpler models of general reference types that nonetheless validate extraordinarily strong equations between stateful programs. To examine this claim, we consider the type of object-oriented counters in a Haskell-like language:

$$\mathsf{Counter} :\equiv \{\mathsf{incr} : \mathsf{IO}\,();\mathsf{read} : \mathsf{IO}\,\mathsf{Int}\}$$

[1] A non-functional approach to compositionality for reference types would be the expression of Reynolds' *capability interpretation* of references [39] in game semantics by Abramsky, Honda, and McCusker [2].

The most obvious implementation of the $\mathsf{Counter}$ interface simply allocates an integer and increments it in memory as follows:

$$\mathsf{posCounter} : \mathsf{IO}\,\mathsf{Counter}$$
$$\mathsf{posCounter} :\equiv \ell \leftarrow \mathsf{alloc}\,0; \mathsf{ret}\,\{\mathsf{incr} \hookrightarrow i \leftarrow \mathsf{get}\,\ell; \mathsf{set}\,\ell\,(i+1), \mathsf{read} \hookrightarrow \mathsf{get}\,\ell\}$$

Another implementation might count *backwards* and then negate the stored value on read using the functorial action map of IO on $\mathsf{neg} : \mathsf{Int} \to \mathsf{Int}$:

$$\mathsf{negCounter} :\equiv \ell \leftarrow \mathsf{alloc}\,0; \mathsf{ret}\,\{\mathsf{incr} \hookrightarrow i \leftarrow \mathsf{get}\,\ell; \mathsf{set}\,\ell\,(i-1), \mathsf{read} \hookrightarrow \mathsf{map}\,\mathsf{neg}\,(\mathsf{get}\,\ell)\}$$

By intuition, the $\mathsf{posCounter}$ and $\mathsf{negCounter}$ implementations of the counter interface should be "observationally equivalent" in the sense that no context of ground type should be able to distinguish them: indeed, even though $\mathsf{negCounter}$ is writing negative numbers to the heap instead of positive numbers, the only way a context can observe the allocated cell is using the read method. The observational equivalence $\mathsf{posCounter} \simeq \mathsf{negCounter}$ is typically proved using a *relational model*, as both Birkedal *et al.* [18, §6.3] and Sterling *et al.* [43] did.

Observational equivalence is not the same as equality, neither in syntax nor semantics. Indeed, typical equational theories of (local or global) dynamic allocation do *not* derive the equation $\vdash \mathsf{posCounter} \equiv \mathsf{negCounter}$, as can be seen easily by means of a countermodel: we have $[\![\mathsf{posCounter}]\!] \neq [\![\mathsf{negCounter}]\!]$ in both the relational models of *op. cit.*, although it is true that $[\![\mathsf{posCounter}]\!]\ R_{\mathsf{IO}\,\mathsf{Counter}}\ [\![\mathsf{negCounter}]\!]$ holds. The observational equivalence $\mathsf{posCounter} \simeq \mathsf{negCounter}$ is deduced in the relational models because the relation on *observations* is discrete.

What distinguishes our ***univalent reference types*** from ordinary reference types is that the former actually derive equations like $\vdash \mathsf{posCounter} \equiv \mathsf{negCounter}$, as shown in Theorem 2.1. We substantiate this equational theory by constructing a model (Theorem 3.25) in univalent foundations [46] in which the equation $\vdash \mathsf{posCounter} \equiv \mathsf{negCounter}$ follows immediately from the univalence principle of the metalanguage. Although it may be possible to validate this equation using parametric models (or less scrupulously by an extensional collapse), our model is the first to get it **for free**.

2 A higher-order language with (univalent) reference types

We begin by giving a description of the syntax and the equational theory of a simple language with references. The language is meant to be *as simple as possible, but no simpler*. In particular, it contains all the problematic constructs (higher-order store, dynamic allocations, *etc.*) that have been traditionally difficult to model denotationally.

2.1 The equational theory of monadic general reference types

While there are many different ways to present programming languages with side effects, for the sake of familiarity we have chosen to focus on a variant of Moggi's *monadic metalanguage* [32].[2] Essentially, this is a simply-typed lambda calculus supplemented with a *strong monad* IO and further equipped with a type of references $\mathsf{IORef}\,\tau$ along with a suite of effectful operations for interacting with references. Like in Haskell, all side effects are confined to the monad; unlike Haskell, general recursion is treated as a side effect.

[2] When developing our denotational semantics in Section 3, we will refine the monadic point of view by passing to an adjoint call-by-push-value decomposition [29].

$$
\begin{array}{llll}
\textit{types} & \tau, \sigma & ::= & \sigma \to \tau \mid \mathsf{IO}\,\tau \mid \mathsf{IORef}\,\tau \mid \ldots \\
\textit{terms} & e, e' & ::= & x \mid \mathsf{rec}\ f\,x\ \mathsf{in}\ e \mid \mathsf{ret}\,e \mid x \leftarrow e;\, e' \mid \mathsf{alloc}\,e \mid \mathsf{get}\,e \mid \mathsf{set}\,e\,e' \mid \mathsf{step} \mid \ldots
\end{array}
$$

$$
\frac{\Gamma \vdash e : \sigma}{\Gamma \vdash \mathsf{alloc}\,e : \mathsf{IO}\,(\mathsf{IORef}\,\sigma)}
\qquad
\frac{\Gamma \vdash e : \mathsf{IORef}\,\sigma}{\Gamma \vdash \mathsf{get}\,e : \mathsf{IO}\,\sigma}
\qquad
\frac{\Gamma \vdash e : \mathsf{IORef}\,\sigma \qquad \Gamma \vdash e' : \sigma}{\Gamma \vdash \mathsf{set}\,e\,e' : \mathsf{IO}\,()}
$$

$$
\frac{}{\Gamma \vdash \mathsf{step} : \mathsf{IO}\,()}
\qquad
\frac{\Gamma, f : \sigma \to \mathsf{IO}\,\tau, x : \sigma \vdash e : \mathsf{IO}\,\tau}{\Gamma \vdash \mathsf{rec}\ f\,x\ \mathsf{in}\ e : \sigma \to \mathsf{IO}\,\tau}
$$

$$
\frac{\Gamma, f : \sigma \to \mathsf{IO}\,\tau, x : \sigma \vdash e : \mathsf{IO}\,\tau \qquad \Gamma \vdash e' : \sigma}{\Gamma \vdash (\mathsf{rec}\ f\,x\ \mathsf{in}\ e)\,e' \equiv \mathsf{step}; [(\mathsf{rec}\ f\,x\ \mathsf{in}\ e)/f, e'/x]e : \mathsf{IO}\,\tau}
$$

$$
\begin{array}{c}
e : \mathsf{IORef}\,\sigma, e' : \sigma \vdash \mathsf{set}\,e\,e'; \mathsf{get}\,e \equiv \mathsf{step}; \mathsf{set}\,e\,e'; \mathsf{ret}\,e' : \mathsf{IO}\,\sigma \\
e : \sigma \vdash (x \leftarrow \mathsf{alloc}\,e; \mathsf{set}\,x\,e) \equiv \mathsf{alloc}\,e : \mathsf{IO}\,(\mathsf{IORef}\,\sigma) \\
e : \mathsf{IORef}\,\sigma, e' : \sigma, e'' : \sigma \vdash \mathsf{set}\,e\,e'; \mathsf{set}\,e\,e'' \equiv \mathsf{set}\,e\,e'' : \mathsf{IO}\,() \\
e : \mathsf{IORef}\,\sigma, e' : \mathsf{IORef}\,\tau \vdash (x \leftarrow \mathsf{get}\,e; y \leftarrow \mathsf{get}\,e'; \mathsf{ret}\,\langle x, y\rangle) \equiv (y \leftarrow \mathsf{get}\,e'; x \leftarrow \mathsf{get}\,e; \mathsf{ret}\,\langle x, y\rangle) : \mathsf{IO}\,(\sigma \times \tau) \\
e : \mathsf{IORef}\,\sigma \vdash (x \leftarrow \mathsf{get}\,e; \mathsf{set}\,e\,x) \equiv \mathsf{get}\,e : \mathsf{IO}\,\sigma \\
e : \mathsf{IORef}\,\sigma, e' : \mathsf{IO}\,\tau \vdash \mathsf{get}\,e; e' \equiv \mathsf{step}; e'
\end{array}
$$

Figure 1 Syntax and selected typing and equational rules for a higher-order monadic language with general reference types. We assume standard notational conventions for monadic programming, *e.g.* writing $e; e'$ for $\_ \leftarrow e; e'$. We assume the standard β/η-equational theory of function and product types, as well as the monadic laws. We also assume that step lies in the *center* [22] of the monad IO, *i.e.* commutes with all monadic operations.

One non-standard aspect of our language bears special attention, namely the nullary side effect $\mathsf{step} : \mathsf{IO}\,()$. This effect can be thought of as the "exhaust" left behind in the equational theory by unfolding any kind of recursively defined construct, including not only the unfolding of recursive functions but also accesses to the heap. In particular, for a given recursive function $g :\equiv \mathsf{rec}\ f\,x\ \mathsf{in}\ e$, we do not have $\vdash g\,e' \equiv [g/f, e'/x]e$ but rather only $\vdash g\,e' \equiv \mathsf{step}; [g/f, e'/x]e$. Likewise, our equational theory does not equate $\vdash (\ell \leftarrow \mathsf{alloc}\,e; \mathsf{get}\,\ell) \equiv \mathsf{ret}\,e$ but rather only $\vdash (\ell \leftarrow \mathsf{alloc}\,e; \mathsf{get}\,\ell) \equiv \mathsf{step}; \mathsf{ret}\,e$. The presence of step in our equational theory is forced by the *guarded* denotational semantics that we will later employ in Section 3 and Theorem 3.25.

2.2 The equational theory of univalent reference types

The equational theory of ***univalent reference types*** strengthens Figure 1 by quotienting under symmetries of the heap, expressed in the two rules depicted in Figure 2.

1. The ALLOCATION PERMUTATION rule states that the order in which references are allocated does not matter; this is a kind of *nominal symmetry* built into the theory of univalent reference types, expressing that the *layout* of the heap is viewed up to isomorphism.

ALLOCATION PERMUTATION

$$\frac{\Gamma \vdash e : \sigma \qquad \Gamma \vdash e' : \tau}{\Gamma \vdash \ell \leftarrow \mathsf{alloc}\, e; \ell' \leftarrow \mathsf{alloc}\, e'; \mathsf{ret}\, \langle \ell, \ell' \rangle \equiv \ell' \leftarrow \mathsf{alloc}\, e'; \ell \leftarrow \mathsf{alloc}\, e; \mathsf{ret}\, \langle \ell, \ell' \rangle : \mathsf{IO}\, (\mathsf{IORef}\, \sigma \times \mathsf{IORef}\, \tau)}$$

REPRESENTATION INDEPENDENCE

$$\frac{\begin{array}{ccc}\Gamma \vdash e : \sigma & \Gamma \vdash f^{+} : \sigma \to \tau & \Gamma \vdash f^{-} : \tau \to \sigma \\ \Gamma, x : \tau \vdash f^{+}(f^{-}x) \equiv x : \tau & & \Gamma, x : \sigma \vdash f^{-}(f^{+}x) \equiv x : \sigma\end{array}}{\Gamma \vdash \ell \leftarrow \mathsf{alloc}\, e; \mathsf{ret}\, \langle \mathsf{get}\, \ell, \mathsf{set}\, \ell \rangle \equiv \ell \leftarrow \mathsf{alloc}\, (f^{+}e); \mathsf{ret}\, \langle \mathsf{map}\, f^{-}\, (\mathsf{get}\, \ell), \mathsf{set}\, \ell \circ f^{+} \rangle : \mathsf{IO}\, (\mathsf{Cell}\, \sigma)}$$

Figure 2 The equational theory of ***univalent reference types***, extending that of Figure 1; we define $\mathsf{Cell}\, \sigma :\equiv \mathsf{IO}\, \sigma \times (\sigma \to \mathsf{IO}\, ())$ to be the "abstract interface" of a reference cell. Here we write $\mathsf{map}\, f : \mathsf{IO}\, A \to \mathsf{IO}\, B$ for the functorial action of IO on a function $f : A \to B$.

2. The REPRESENTATION INDEPENDENCE rule states that the *observable interface* of a reference cell is invariant under isomorphisms of that cell's contents.

The ALLOCATION PERMUTATION rule is common to theories of local dynamic allocation, but less common in theories of global dynamic allocation. The REPRESENTATION INDEPENDENCE rule is, however, a new feature of univalent reference types that goes beyond existing local theories of dynamic allocation: as we have discussed in Section 1.4, such a law typically holds up to observational equivalence but almost never "on the nose" at higher type. It is therefore worth going into more detail.

The idea of REPRESENTATION INDEPENDENCE is that allocating a cell of type σ and then only interacting with it by means of its $(\mathsf{get}, \mathsf{set})$ methods should be the same as allocating a cell of a different type τ and interacting with it by conjugating its $(\mathsf{get}, \mathsf{set})$ interface by an isomorphism $e : \sigma \cong \tau$. In particular, it is allowed that $\sigma \equiv \tau$ and $e : \sigma \cong \sigma$ be nonetheless a non-trivial automorphism: and so we may derive from REPRESENTATION INDEPENDENCE our case study involving imperative counters that count forward and backwards.

▶ **Theorem 2.1.** *Let* $\mathsf{Counter}$, $\mathsf{posCounter}$, *and* $\mathsf{negCounter}$ *be as in Section 1.4:*

$$\begin{aligned}
&\mathsf{Counter} :\equiv \{\mathsf{incr} : \mathsf{IO}\, (); \mathsf{read} : \mathsf{IO}\, \mathsf{Int}\}\\
&\mathsf{posCounter} :\equiv \ell \leftarrow \mathsf{alloc}\, 0; \mathsf{ret}\, \{\mathsf{incr} \hookrightarrow i \leftarrow \mathsf{get}\, \ell; \mathsf{set}\, \ell\, (i+1), \mathsf{read} \hookrightarrow \mathsf{get}\, \ell\}\\
&\mathsf{negCounter} :\equiv \ell \leftarrow \mathsf{alloc}\, 0; \mathsf{ret}\, \{\mathsf{incr} \hookrightarrow i \leftarrow \mathsf{get}\, \ell; \mathsf{set}\, \ell\, (i-1), \mathsf{read} \hookrightarrow \mathsf{map}\, \mathsf{neg}\, (\mathsf{get}\, \ell)\}
\end{aligned}$$

We may derive $\vdash \mathsf{posCounter} \equiv \mathsf{negCounter} : \mathsf{Counter}$.

Proof. The function $\mathsf{neg} : \mathsf{Int} \to \mathsf{Int}$ sending an integer to its negation is a self-dual automorphism; we therefore calculate from left to right.

$\ell \leftarrow \mathsf{alloc}\, 0; \mathsf{ret}\, \{\mathsf{incr} \hookrightarrow i \leftarrow \mathsf{get}\, \ell; \mathsf{set}\, \ell\, (i+1), \mathsf{read} \hookrightarrow \mathsf{get}\, \ell\}$

by REPRESENTATION INDEPENDENCE

$\equiv \ell \leftarrow \mathsf{alloc}\, (\mathsf{neg}\, 0); \mathsf{ret}\, \{\mathsf{incr} \hookrightarrow i \leftarrow \mathsf{map}\, \mathsf{neg}\, (\mathsf{get}\, \ell); \mathsf{set}\, \ell\, (\mathsf{neg}\, (i+1)), \mathsf{read} \hookrightarrow \mathsf{map}\, \mathsf{neg}\, (\mathsf{get}\, \ell)\}$

by simplification and $\mathsf{neg}\, 0 \equiv 0$

$\equiv \ell \leftarrow \mathsf{alloc}\, 0; \mathsf{ret}\, \{\mathsf{incr} \hookrightarrow i \leftarrow \mathsf{get}\, \ell; \mathsf{set}\, \ell\, (\mathsf{neg}\, (\mathsf{neg}\, i + 1)), \mathsf{read} \hookrightarrow \mathsf{map}\, \mathsf{neg}\, (\mathsf{get}\, \ell)\}$

by $\mathsf{neg}\, (\mathsf{neg}\, i + 1) \equiv \mathsf{neg}\, (\mathsf{neg}\, i) + \mathsf{neg}\, 1 \equiv i - 1$

$\equiv \ell \leftarrow \mathsf{alloc}\, 0; \mathsf{ret}\, \{\mathsf{incr} \hookrightarrow i \leftarrow \mathsf{get}\, \ell; \mathsf{set}\, \ell\, (i-1), \mathsf{read} \hookrightarrow \mathsf{map}\, \mathsf{neg}\, (\mathsf{get}\, \ell)\}$

Thus we have $\mathsf{posCounter} \equiv \mathsf{negCounter}$. ◀

3 Denotational semantics in univalent foundations

We now turn to the construction of a model of univalent reference types. At the coarsest level, this model follows the standard template for a model with mutable state: types are interpreted by covariant presheaves on a certain category of *worlds* with each world describing the collection of references available and the (semantic) type associated to each. The type of references $\mathsf{IORef}\,\tau$ assigns each world to the collection of locations of appropriate type while the monad IO is then interpreted by a certain *store-passing* monad.

This simple picture is quickly complicated by the need to model general store: semantic types must reference to worlds which in turn reference semantic types. This naturally leads us to synthetic guarded domain theory (SGDT) in order to cope with the circularity. This alone, however, is insufficient. While SGDT allows us to define the category of worlds, the resulting solution is a *large type*—at least the size of the universe of semantic types. This becomes a problem when it comes time to model the state monad, which must quantify over all possible worlds for its input and return a new world for its output. To model these large products and sums of worlds, we will base our model on an *impredicative* universe: impredicativity implies that the category of covariant presheaves on our large category of worlds is (locally) cartesian closed and supports all the structure of our language.

We present some of the prerequisites for our model in Section 3.1. In Section 3.2 we construct the model of univalent reference types.

3.1 Univalent impredicative synthetic guarded domain theory

We work informally in the language of homotopy type theory [40, 46]; in this section, we briefly describe some of our preferred conventions. When we speak of "existence", we shall always mean *mere* existence. Categories are always assumed to be univalent 1-categories; given a category $\mathcal{X}$, we will write $|\mathcal{X}|$ for its underlying 1-type of objects. Rather than fixing a global hierarchy of universes, we assume universes locally where needed. In this paper, all universes are assumed to be univalent; when we wish to assume that a universe is closed under the connectives of Martin-Löf type theory (dependent products, dependent sums, finite coproducts, W-types, *etc.*) we will refer to it as a ***Martin-Löf universe***. We will not belabor the difference between codes and types.

3.1.1 Impredicative subuniverses in univalent foundations

Recall that a type A is called ***$\mathcal{U}$-small*** if and only if there exists a (necessarily unique) code $\hat{A} : \mathcal{U}$ together with an equivalence $[\hat{A}] \simeq A$, and a family is $\mathcal{U}$-small when each of its fibers are. A ***reflection*** of A in a universe $\mathcal{U}$ is, by contrast, defined to be a (necessarily unique) function $\eta : A \to A_{\mathcal{U}}$ with $A_{\mathcal{U}} \in \mathcal{U}$ such for any type $C \in \mathcal{U}$, the precomposition map $C^{\eta} : C^{A_{\mathcal{U}}} \to C^{A}$ is an equivalence. When a reflection of A in $\mathcal{U}$ exists (necessarily uniquely), we shall say that A is reflected in $\mathcal{U}$.

A ***subuniverse*** of a universe $\mathcal{U}$ is defined to be a dependent type $A : \mathcal{U} \vdash PA$ *type* such that each PA is a proposition. We may write $\mathcal{U}_P$ for the universe $\sum_{A:\mathcal{U}} PA$ obtained by restricting $\mathcal{U}$ to the elements satisfying P. We will frequently abuse notation implicitly identifying the predicate coding a subuniverse with its comprehension as an actual type. A subuniverse $\mathcal{S} \subseteq \mathcal{U}$ is said to be ***reflective*** if every $A : \mathcal{U}$ is reflected in $\mathcal{S}$. A subuniverse of $\mathcal{U}$ is said to be "small" when its comprehension as a type is $\mathcal{U}$-small.

Let $\mathcal{U}$ be a universe closed under dependent products. A subuniverse $\mathcal{S} \subseteq \mathcal{U}$ is said to be a ***dependent exponential ideal*** if for every $A : \mathcal{U}$ and $B : A \to \mathcal{S}$, the dependent product $\prod_{x:A} Bx$ lies in $\mathcal{S}$. An ***impredicative subuniverse*** of $\mathcal{U}$ is defined to be a small, dependent exponential ideal $\mathcal{S} \subseteq \mathcal{U}$. It is proved by Rijke, Shulman, and Spitters [41] that any reflective subuniverse of $\mathcal{U}$ is a dependent exponential ideal of $\mathcal{U}$. We will refer to a subuniverse $\mathcal{S}$ such that $\mathsf{Set}_{\mathcal{S}}$ is impredicative in $\mathcal{U}$ as ***set-impredicative***; we will refer to $\mathcal{S} \subseteq \mathcal{U}$ as ***set-reflective*** when $\mathsf{Set}_{\mathcal{S}}$ is reflective in $\mathcal{U}$. Under suitable assumptions, these two conditions are in fact equivalent:

▶ **Theorem 3.1.** *A small Σ-closed subuniverse $\mathcal{S}$ of a Martin-Löf universe $\mathcal{U}$ is set-impredicative and closed under identity types if and only if it is set-reflective.*

Proof. This can be shown using the methods of Awodey, Frey, and Speight [13]. ◀

By virtue of Theorem 3.1, we see that small reflective subuniverses are just another presentation of the impredicative universes that appear in the Calculus of Constructions.

3.1.2 The Hofmann–Streicher universe

Let $\mathcal{S}$ be a small subuniverse of a Martin-Löf universe $\mathcal{U}$, and let $\mathcal{X}$ be a $\mathcal{U}$-small category; we can define the ***Hofmann–Streicher lifting*** [24, 12] of $\mathsf{Set}_{\mathcal{S}}$ as co-presheaf of 1-types on $\mathcal{X}$. Formally, this means constructing a functor from the 1-category $\mathcal{X}$ to the (2,1)-category $\mathsf{1Type}_{\mathcal{U}}$ of 1-types in $\mathcal{U}$; thus we depend technically on the account of bicategories in univalent foundations due to Ahrens *et al.* [4].[3]

▶ Remark 3.2. The purpose of introducing the Hofmann–Streicher lifting in such detail is give some structure to the otherwise bewildering Construction 3.4, which plays a crucial technical role in the definition of univalent reference types.

▶ **Construction 3.3** (The Hofmann–Streicher lifting)**.** Let $\mathcal{S}$ be a small subuniverse of a Martin-Löf universe $\mathcal{U}$, and let $\mathcal{X}$ be a $\mathcal{U}$-small category. We may define a 2-functor $\lfloor \mathsf{Set}_{\mathcal{S}} \rfloor : \mathcal{X} \to \mathsf{1Type}_{\mathcal{U}}$ called the ***Hofmann–Streicher lifting*** of $\mathsf{Set}_{\mathcal{S}}$ as follows:

$$\lfloor \mathsf{Set}_{\mathcal{S}} \rfloor U :\equiv |\mathsf{Fun}(U/\mathcal{X}, \mathsf{Set}_{\mathcal{S}})|$$
$$\lfloor \mathsf{Set}_{\mathcal{S}} \rfloor (f : U \to V)\,(E : U/\mathcal{X} \to \mathsf{Set}_{\mathcal{S}}) :\equiv E \circ (f/\mathcal{X})$$

When $\mathcal{X}$ is viewed as a 2-category, the 2-cells are given by identifications. Thus the 2-functoriality of $\lfloor \mathsf{Set}_{\mathcal{S}} \rfloor$ and all related coherences are defined by path induction.

▶ **Construction 3.4** (Restricting co-presheaves)**.** Let $\mathcal{S}$ be a small subuniverse of a Martin-Löf universe $\mathcal{U}$, and let $\mathcal{X}$ be a $\mathcal{U}$-small category. Every co-presheaf $E : \mathcal{X} \to \mathsf{Set}_{\mathcal{S}}$ determines a global element $\mathbf{1} \to \lfloor \mathsf{Set}_{\mathcal{S}} \rfloor$ of the Hofmann–Streicher universe; in particular, we may define $\lceil E \rceil_U : \lfloor \mathsf{Set}_{\mathcal{S}} \rfloor U$ natural in $U \in \mathcal{X}$ by setting $\lceil E \rceil_U\,(f : U \to V) :\equiv EV$.

3.1.3 (Higher) synthetic guarded domain theory

We adapt Birkedal *et al.*'s formulation [19] of dependently typed guarded recursion to the setting of homotopy type theory. In particular, we introduce a new syntactic sort of *delayed substitutions* $\vdash \xi \rightsquigarrow \Xi$ simultaneously with a new type former $\vdash \blacktriangleright[\xi].A$ *type* called the ***later***

[3] We differ from the conventions of Ahrens *et al.* [4]: we will say "2-category" to mean *univalent bicategory* in the sense of *op. cit.*, as we are not at all concerned with the strict notions considered there. Therefore, a "(2,1)-category" in our sense refers to a 2-category whose 2-cells are given by identifications.

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

$$\frac{f : \blacktriangleright A \to A}{\mathsf{fix}_{\blacktriangleright}\, f : A} \qquad \frac{f : \blacktriangleright A \to A}{\mathsf{fix}_{\blacktriangleright}\, f \equiv f\,(\mathsf{next}\,(\mathsf{fix}_{\blacktriangleright}\, f))}$$

Figure 3 Summary of delayed substitutions and the later modality; there are a number of equational rules governing the delayed substitutions, *e.g.* $\blacktriangleright[\xi, x \leftarrow a].A \equiv \blacktriangleright[\xi].A$ for any A in which x does not appear; we also assume $(\blacktriangleright[\xi].a = b) \simeq (\mathsf{next}[\xi].a = \mathsf{next}[\xi].b)$, making $\blacktriangleright$ left exact. We will write $\blacktriangleright A$ and $\mathsf{next}\,a$ for $\blacktriangleright[\cdot].A$ and $\mathsf{next}[\cdot].a$ respectively. For the remaining rules, we refer the reader to the description of Bizjak and Møgelberg [20].

modality,[4] whose introduction form is written $\mathsf{next}[\xi].a$; we summarize the rules for the later modality in Figure 3. The *raison d'être* for the later modality is to form ***guarded fixed points***: in particular, if we have $f : \blacktriangleright A \to A$, there is a *unique* element $\mathsf{fix}_{\blacktriangleright} f : A$ such that $f\,(\mathsf{next}\,(\mathsf{fix}_{\blacktriangleright} f)) = \mathsf{fix}_{\blacktriangleright} f$. In particular, this gives unique fixed points for any function $f : A \to A$ factoring on the left through $\mathsf{next} : A \to \blacktriangleright A$.

▶ **Definition 3.5.** *A **guarded (n-)domain** is an (n-)type A equipped with the structure of a $\blacktriangleright$-algebra,* i.e. *a function $\vartheta_A : \blacktriangleright A \to A$.*

We will refer to a (sub)universe closed under later modalities as a ***guarded (sub)universe***. For any universe $\mathcal{S}$, we may consider the category $\mathsf{0Dom}_{\mathcal{S}}$ of ***guarded 0-domains*** in $\mathcal{S}$, *i.e.* sets $A : \mathsf{Set}_{\mathcal{S}}$ equipped with a mapping $\vartheta_A : \blacktriangleright A \to A$.

▶ **Lemma 3.6.** *If $\mathcal{S}$ is a guarded universe closed under binary coproducts, then the forgetful functor $\mathsf{R} : \mathsf{0Dom}_{\mathcal{S}} \to \mathsf{Set}_{\mathcal{S}}$ has a left adjoint $\mathsf{L} : \mathsf{Set}_{\mathcal{S}} \to \mathsf{0Dom}_{\mathcal{S}}$.*

Proof. We define $\mathsf{L}A$ by solving the domain equation $\mathsf{L}A \cong A + \blacktriangleright \mathsf{L}A$ via the following guarded fixed point construction in $\mathsf{Set}_{\mathcal{S}}$, using both guarded structure and binary coproducts:

$$\mathsf{L}A :\equiv \mathsf{fix}_{\blacktriangleright}(\lambda X : \blacktriangleright \mathsf{Set}_{\mathcal{S}}.A + \blacktriangleright[Y \leftarrow X].Y)$$ ◀

▶ **Notation 3.7.** We will write $\mathsf{now} : A \to \mathsf{RL}A$ for the unit of the adjunction $\mathsf{L} \dashv \mathsf{R}$.

▶ **Lemma 3.8** (Later modality in presheaves). *Given a guarded set-reflective small subuniverse $\mathcal{S} \subseteq \mathcal{U}$ and a $\mathcal{U}$-small category $\mathfrak{X}$, the later modality from $\mathcal{S}$ lifts (with all its operations) into $\mathsf{Fun}(\mathfrak{X}, \mathsf{Set}_{\mathcal{S}})$ pointwise,* i.e. *for any $A \in \mathsf{Fun}(\mathfrak{X}, \mathsf{Set}_{\mathcal{S}})$ we may define $(\blacktriangleright A)U :\equiv \blacktriangleright(AU)$.*

3.2 Models of univalent general reference types

To construct our model of higher-order store (Section 3.2.3), we must construct a suitable category of recursively defined semantic worlds (Section 3.2.1) whose co-presheaves admit reference types (Construction 3.13) and a strong monad for higher-order store (Section 3.2.2).

[4] The "later modality" is *not* a modality in the sense of Rijke, Shulman, and Spitters [41], but rather in the older and more general sense of modality in type theory or logic.

3.2.1 Worlds as univalent heap configurations

Let Inj be the category of finite sets and embeddings; by univalence, any two equipollent finite sets are identified. We now define the basic elements of worlds *qua* heap configurations.

▶ **Definition 3.9** (The displayed category of families). *For any 1-type X, we define the displayed category [5] IFam_X of Inj-**families in** X over Inj as follows:*
1. *over a finite set $I : \mathsf{Inj}$, a displayed object of IFam_X is a function $\partial_I : I \to X$;*
2. *over a function $f : I \to J$ between finite sets, a displayed morphism from ∂_I to ∂_J is a path $\partial_f : \partial_J \circ f = \partial_I$ in $I \to X$.*

▶ **Definition 3.10** (The category of bags). *For any 1-type X, the category of Inj-**bags in** X is defined to be the total category $\mathsf{IBag}_X :\equiv \int_{\mathsf{Inj}} \mathsf{IFam}_X$ of the displayed category of finite families in X. We will write $U \equiv (|U|, \partial_U)$ for an object of IBag_X.*

▶ **Definition 3.11.** *For a universe $\mathcal{S}$, we define the category $\mathcal{C}_{\mathcal{S}}$ of **worlds** simultaneously with its category of $\mathsf{Set}_{\mathcal{S}}$-valued co-presheaves on $\mathcal{C}_{\mathcal{S}}$ to be the unique solution to the guarded recursive domain equation $\mathcal{C}_{\mathcal{S}} = \mathsf{IBag}_{\blacktriangleright|\mathsf{Fun}(\mathcal{C}_{\mathcal{S}}, \mathsf{Set}_{\mathcal{S}})|}$.*

We shall require the following technical observation:

▶ **Lemma 3.12** (Structure identity principle for presheaves). *Let $\mathcal{S}$ be a universe and let $\mathcal{X}$ be a category; for any $A, B : \mathsf{Fun}(\mathcal{X}, \mathsf{Set}_{\mathcal{S}})$, let $A \cong B$ be the type of natural isomorphisms between presheaves. Then the canonical map $A = B \to A \cong B$ is an equivalence.*

We now come to the construction of the univalent reference type constructor.

▶ **Construction 3.13** (Univalent references). Let $\mathcal{S}$ be a small, guarded, Σ-closed set-reflective subuniverse of a guarded Martin-Löf universe $\mathcal{U}$ containing Inj. We define the ***univalent reference type constructor*** as a mapping $\mathsf{IORef} : |\mathsf{Fun}(\mathcal{C}_{\mathcal{S}}, \mathsf{Set}_{\mathcal{S}})| \to |\mathsf{Fun}(\mathcal{C}_{\mathcal{S}}, \mathsf{Set}_{\mathcal{S}})|$:

$$\mathsf{IORef} : |\mathsf{Fun}(\mathcal{C}_{\mathcal{S}}, \mathsf{Set}_{\mathcal{S}})| \to |\mathsf{Fun}(\mathcal{C}_{\mathcal{S}}, \mathsf{Set}_{\mathcal{S}})|$$
$$\mathsf{IORef}\, A\, U :\equiv \textstyle\sum_{\ell:|U|} \blacktriangleright[X \leftarrow \partial_U \ell].\ \lceil X \rceil_U = \lceil A \rceil_U$$

Above, we have used the $\lceil - \rceil$ operator from Construction 3.4. We define the functorial action of $\mathsf{IORef}\, A$ on $f : U \to V$ by path induction on the identification $\partial_f : \partial_V \circ |f| = \partial_U$:

$$\mathsf{IORef}\, A\, (|f|, \mathsf{refl})\, (\ell, \phi) :\equiv (|f|\ell, \mathsf{next}[X \leftarrow \partial_V(|f|\ell), \psi \leftarrow \phi].\ \mathsf{ap}_{\lfloor \mathsf{Set}_{\mathcal{S}} \rfloor f} \psi)$$

Proof. That the identification $\lceil X \rceil_U = \lceil A \rceil_U$ is $\mathcal{S}$-small follows from Lemma 3.12, using the fact that $U/\mathcal{C}_{\mathcal{S}}$ is $\mathcal{U}$-small because $\mathcal{S}$ is assumed $\mathcal{U}$-small. ◀

3.2.2 A strong monad for general store

Rather than constructing the monad for general store all at once by hand, we take a more bite-sized approach by decomposing it into a simpler call-by-push-value adjunction following Levy [29]. In fact, we go quite a bit further than this and decompose the call-by-push-value adjunction itself into three separate and simpler adjunctions; the advantage of our decomposition is that it reveals the simple and elegant source of the admittedly complex explicit constructions of *op. cit.* All these adjunctions will be suitably *enriched* so as to give rise to a strong monad. To get started, we will first require the concept of a ***heaplet***, which is the valuation of a heap configuration at a particular world, assigning each specified location to an element of the prescribed semantic type at that world.

▶ **Construction 3.14** (The heaplet distributor)**.** Let $\mathcal{S}$ be guarded universe closed under finite products. We may define a distributor $\mathcal{H}_{\mathcal{S}} : \mathcal{C}_{\mathcal{S}}^{\mathsf{op}} \times \mathcal{C}_{\mathcal{S}} \to \mathsf{Set}_{\mathcal{S}}$ like so:

$$
\begin{aligned}
&\mathcal{H}_{\mathcal{S}} : \mathcal{C}_{\mathcal{S}}^{\mathsf{op}} \times \mathcal{C}_{\mathcal{S}} \to \mathsf{Set}_{\mathcal{S}} \\
&\mathcal{H}_{\mathcal{S}}\,(U, V) :\equiv \textstyle\prod_{\ell:|U|} \vartheta_{|\mathsf{Fun}(\mathcal{C}_{\mathcal{S}},\mathsf{Set}_{\mathcal{S}})|}\,(\partial_U \ell)\,V
\end{aligned}
$$

Then we will write $\widetilde{\mathcal{H}_{\mathcal{S}}}$ for the dependent sum $\sum_{U:|\mathcal{C}_{\mathcal{S}}|} \mathcal{H}_{\mathcal{S}}\,U\,U$ classifying ***heaps***. We will write $\pi_{\mathcal{H}} : \widetilde{\mathcal{H}_{\mathcal{S}}} \to |\mathcal{C}_{\mathcal{S}}|$ for the first projection of a packed heap; given $H : \widetilde{\mathcal{H}_{\mathcal{S}}}$ and $\ell : |\pi_{\mathcal{H}} H|$, we will write $H @ \ell : \blacktriangleright[X \leftarrow \partial_{\pi_{\mathcal{H}} H}\ell]X(\pi_{\mathcal{H}} H)$ for the element stored by H at location ℓ.

Presheaf categories and unenriched adjunctions

Let $\mathcal{S}$ be a guarded universe closed under finite products. As $\widetilde{\mathcal{H}_{\mathcal{S}}}$ is a 1-type, we can equally well view it as a category whose hom sets are given by identity types, *i.e.* a groupoid. From this point of view, the projection $\pi_{\mathcal{H}} : \widetilde{\mathcal{H}_{\mathcal{S}}} \to |\mathcal{C}_{\mathcal{S}}|$ extends to functors $\pi_{\mathcal{H}} : \widetilde{\mathcal{H}_{\mathcal{S}}} \to \mathcal{C}_{\mathcal{S}}$ and $\bar{\pi}_{\mathcal{H}} : \widetilde{\mathcal{H}_{\mathcal{S}}} \cong \widetilde{\mathcal{H}_{\mathcal{S}}}^{\mathsf{op}} \to \mathcal{C}_{\mathcal{S}}^{\mathsf{op}}$. We will use these projections to construct a network of adjunctions between the following presheaf categories:

$$
\begin{aligned}
&\mathcal{P}_{\mathcal{S}} :\equiv \mathsf{Fun}(\mathcal{C}_{\mathcal{S}}, \mathsf{Set}_{\mathcal{S}}) && \mathcal{N}_{\mathcal{S}} :\equiv \mathsf{Fun}(\mathcal{C}_{\mathcal{S}}^{\mathsf{op}}, \mathsf{0Dom}_{\mathcal{S}}) \\
&\bar{\mathcal{P}}_{\mathcal{S}} :\equiv \mathsf{Fun}(\mathcal{C}_{\mathcal{S}}^{\mathsf{op}}, \mathsf{Set}_{\mathcal{S}}) && \mathcal{Q}_{\mathcal{S}} :\equiv \mathsf{Fun}(\widetilde{\mathcal{H}_{\mathcal{S}}}, \mathsf{Set}_{\mathcal{S}})
\end{aligned}
$$

▶ **Exegesis 3.15.** $\mathcal{P}_{\mathcal{S}}$ is the category on which our higher-order state monad is defined; this monad arises from a call-by-push-value adjunction [29] in which $\mathcal{P}_{\mathcal{S}}$ is the category of “value types and pure functions” and $\mathcal{N}_{\mathcal{S}}$ is the category of “computation types and stacks”. A computation type differs from a value type in two ways, as it is both *contravariantly* indexed in worlds and valued in 0-domains rather than sets. We will treat these differences modularly by factoring the adjunction $\mathsf{F} \dashv \mathsf{U} : \mathcal{N}_{\mathcal{S}} \to \mathcal{P}_{\mathcal{S}}$ through further adjunctions. Our first adjoint resolution, to deal strictly with variance, is described in Lemma 3.16; later on in Lemma 3.19, we will lift the adjunction between sets and 0-domains to the world of presheaves.

▶ **Lemma 3.16.** *Let $\mathcal{S}$ be a small, set-reflective, guarded subuniverse of a guarded Martin-Löf universe $\mathcal{U}$ containing* Inj*. Then the unenriched base change functors $\Delta_{\pi_{\mathcal{H}}} : \mathcal{P}_{\mathcal{S}} \to \mathcal{Q}_{\mathcal{S}}$ and $\Delta_{\bar{\pi}_{\mathcal{H}}} : \bar{\mathcal{P}}_{\mathcal{S}} \to \mathcal{Q}_{\mathcal{S}}$ has left and right adjoints $\exists_{\bar{\pi}_{\mathcal{H}}} \dashv \Delta_{\bar{\pi}_{\mathcal{H}}}$ and $\Delta_{\pi_{\mathcal{H}}} \dashv \forall_{\pi_{\mathcal{H}}}$ respectively.*

Proof. As $\widetilde{\mathcal{H}_{\mathcal{S}}}$ is both discrete and $\mathcal{U}$-small, the Kan extensions exist because $\mathsf{Set}_{\mathcal{S}}$ has all $\mathcal{U}$-small coproducts and products as a reflective subuniverse of $\mathcal{U}$. ◀

The unenriched adjunctions of Lemma 3.16 can be computed on objects as follows, where $\bigcirc : \mathcal{U} \to \mathsf{Set}_{\mathcal{S}}$ is the assumed reflection:

$$
\begin{aligned}
&\exists_{\bar{\pi}_{\mathcal{H}}} A\, U = \bigcirc \textstyle\sum_{H:\widetilde{\mathcal{H}_{\mathcal{S}}}} \sum_{f:\hom_{\mathcal{C}_{\mathcal{S}}^{\mathsf{op}}}(\pi_{\mathcal{H}} H, U)} AH \\
&\forall_{\pi_{\mathcal{H}}} A\, U = \textstyle\prod_{H:\widetilde{\mathcal{H}_{\mathcal{S}}}} \prod_{f:\hom_{\mathcal{C}_{\mathcal{S}}}(U, \pi_{\mathcal{H}} H)} AH
\end{aligned}
$$

We draw attention to the fact that codomain of $\exists_{\bar{\pi}_{\mathcal{H}}}$ is $\bar{\mathcal{P}}_{\mathcal{S}}$ while the codomain of $\forall_{\bar{\pi}_{\mathcal{H}}}$ is $\mathcal{P}_{\mathcal{S}}$, hence the appearance of $\hom_{\mathcal{C}_{\mathcal{S}}^{\mathsf{op}}}$ in the former and $\hom_{\mathcal{C}_{\mathcal{S}}}$ in the latter.

Enrichments and enriched adjunctions

Let $\mathcal{S}$ be a guarded universe closed under finite products. We now impose a common enrichment on $\mathcal{P}_{\mathcal{S}}, \bar{\mathcal{P}}_{\mathcal{S}}, \mathcal{N}_{\mathcal{S}}, \mathcal{Q}_{\mathcal{S}}$ so as to lift Lemma 3.16 to the enriched level, making all these categories *locally indexed* in $\mathcal{P}_{\mathcal{S}}$ in the sense of [29]. Given a co-presheaf $\Gamma \in \mathcal{P}_{\mathcal{S}}$, we will write $\pi_{\Gamma} : \tilde{\Gamma} \to \mathcal{C}_{\mathcal{S}}^{\mathsf{op}}$ for the discrete cartesian fibration corresponding to Γ. With this in hand, we impose the following additional notations:

$$\mathcal{P}^{\Gamma}_{\mathcal{S}} :\equiv \mathsf{Fun}(\tilde{\Gamma}^{\mathsf{op}}, \mathsf{Set}_{\mathcal{S}}) \qquad \mathcal{N}^{\Gamma}_{\mathcal{S}} :\equiv \mathsf{Fun}(\tilde{\Gamma}, \widetilde{\mathsf{0Dom}_{\mathcal{S}}})$$
$$\bar{\mathcal{P}}^{\Gamma}_{\mathcal{S}} :\equiv \mathsf{Fun}(\tilde{\Gamma}, \mathsf{Set}_{\mathcal{S}}) \qquad \mathcal{Q}^{\Gamma}_{\mathcal{S}} :\equiv \mathsf{Fun}(\tilde{\Gamma}^{\mathsf{op}} \times_{\mathcal{C}_{\mathcal{S}}} \widetilde{\mathcal{H}_{\mathcal{S}}}, \mathsf{Set}_{\mathcal{S}})$$

Above, we note that $\mathcal{P}^{\Gamma}_{\mathcal{S}}$ is equivalent to the slice $\mathcal{P}_{\mathcal{S}}/\Gamma$; in the definition of $\mathcal{Q}^{\Gamma}_{\mathcal{S}}$, the expression $\tilde{\Gamma}^{\mathsf{op}} \times_{\mathcal{C}_{\mathcal{S}}} \widetilde{\mathcal{H}_{\mathcal{S}}}$ refers to the pullback of the span $\{\tilde{\Gamma}^{\mathsf{op}} \xrightarrow{\pi_{\Gamma}^{\mathsf{op}}} \mathcal{C}_{\mathcal{S}} \xleftarrow{\pi_{\mathcal{H}}} \widetilde{\mathcal{H}_{\mathcal{S}}}\}$. We have the following base change functors for any co-presheaf $\Gamma \in \mathcal{P}_{\mathcal{S}}$:

$$\Delta_{\Gamma} : \mathcal{P}_{\mathcal{S}} \to \mathcal{P}^{\Gamma}_{\mathcal{S}} \qquad \Delta_{\Gamma} : \mathcal{N}_{\mathcal{S}} \to \mathcal{N}^{\Gamma}_{\mathcal{S}}$$
$$\Delta_{\Gamma} A\,(U, \gamma) :\equiv AU \qquad \Delta_{\Gamma}\, X\,(U, \Gamma) :\equiv XU$$

$$\Delta_{\Gamma} : \bar{\mathcal{P}}_{\mathcal{S}} \to \bar{\mathcal{P}}^{\Gamma}_{\mathcal{S}} \qquad \Delta_{\Gamma} : \mathcal{Q}_{\mathcal{S}} \to \mathcal{Q}^{\Gamma}_{\mathcal{S}}$$
$$\Delta_{\Gamma} X\,(U, \gamma) :\equiv XU \qquad \Delta_{\Gamma}\, A\,(U, \Gamma, H) :\equiv A(U, H)$$

► **Construction 3.17** (Enrichments). We extend $\mathcal{P}_{\mathcal{S}}$, $\bar{\mathcal{P}}_{\mathcal{S}}$, $\mathcal{N}_{\mathcal{S}}$, and $\mathcal{Q}_{\mathcal{S}}$ to $\hat{\mathcal{P}}_{\mathcal{S}}$-enriched categories $\underline{\mathcal{P}_{\mathcal{S}}}$, $\underline{\bar{\mathcal{P}}_{\mathcal{S}}}$, $\underline{\mathcal{N}_{\mathcal{S}}}$, and $\underline{\mathcal{Q}_{\mathcal{S}}}$ respectively, regarding $\hat{\mathcal{P}}_{\mathcal{S}}$ with its *cartesian* monoidal structure:

$$\hom_{\underline{\mathcal{P}_{\mathcal{S}}}}(A, B)_{\Gamma} :\equiv \hom_{\mathcal{P}^{\Gamma}_{\mathcal{S}}}(\Delta_{\Gamma} A, \Delta_{\Gamma} B) \qquad \hom_{\underline{\mathcal{N}_{\mathcal{S}}}}(X, Y)_{\Gamma} :\equiv \hom_{\mathcal{N}^{\Gamma}_{\mathcal{S}}}(\Delta_{\Gamma} X, \Delta_{\Gamma} Y)$$
$$\hom_{\underline{\bar{\mathcal{P}}_{\mathcal{S}}}}(X, Y)_{\Gamma} :\equiv \hom_{\bar{\mathcal{P}}^{\Gamma}_{\mathcal{S}}}(\Delta_{\Gamma} X, \Delta_{\Gamma} Y) \qquad \hom_{\underline{\mathcal{Q}_{\mathcal{S}}}}(A, B)_{\Gamma} :\equiv \hom_{\mathcal{Q}^{\Gamma}_{\mathcal{S}}}(\Delta_{\Gamma} A, \Delta_{\Gamma} B)$$

These enrichments agree with those given by Levy [29] in terms of dinatural transformations as one can see using the formula for a natural transformation as an end. The purpose of imposing these enrichments was to be able to state Lemmas 3.18 and 3.19 below.

► **Lemma 3.18.** *Under the assumptions of Lemma 3.16, the unenriched adjunctions* $\Delta_{\pi_{\mathcal{H}}} \dashv \forall_{\pi_{\mathcal{H}}}$ *and* $\exists_{\bar{\pi}_{\mathcal{H}}} \dashv \Delta_{\bar{\pi}_{\mathcal{H}}}$ *extend to* $\hat{\mathcal{P}}_{\mathcal{S}}$*-enriched adjunctions* $\underline{\Delta}_{\pi_{\mathcal{H}}} \dashv \underline{\forall}_{\pi_{\mathcal{H}}}$ *and* $\underline{\exists}_{\bar{\pi}_{\mathcal{H}}} \dashv \underline{\Delta}_{\bar{\pi}_{\mathcal{H}}}$.

► **Lemma 3.19.** *Let* $\mathcal{S}$ *be a guarded universe closed under binary coproducts. Then the adjunction* $\mathsf{L} \dashv \mathsf{R} : \mathsf{0Dom}_{\mathcal{S}} \to \mathsf{Set}_{\mathcal{S}}$ *between sets and guarded 0-domains (Lemma 3.6) can be lifted pointwise to an enriched adjunction* $\mathsf{L} \dashv \mathsf{R} : \underline{\mathcal{N}_{\mathcal{S}}} \to \underline{\bar{\mathcal{P}}_{\mathcal{S}}}$.

The call-by-push-value adjunction and resulting strong monad

Let $\mathcal{S}$ be a small, set-reflective, guarded subuniverse of a guarded Martin-Löf universe $\mathcal{U}$ containing Inj. We can compose the enriched adjunctions obtained in Lemma 3.18 to obtain a single enriched adjunction $\mathsf{F} \dashv \mathsf{U} : \underline{\mathcal{N}_{\mathcal{S}}} \to \underline{\mathcal{P}_{\mathcal{S}}}$, setting $\mathsf{F} :\equiv \mathsf{L} \circ \underline{\exists}_{\bar{\pi}_{\mathcal{H}}} \circ \underline{\Delta}_{\pi_{\mathcal{H}}}$ and $\mathsf{U} :\equiv \underline{\forall}_{\pi_{\mathcal{H}}} \circ \underline{\Delta}_{\bar{\pi}_{\mathcal{H}}} \circ \mathsf{R}$ as depicted in Figure 4. We will write ret for the unit of this adjunction. Our adjunction is an adjoint decomposition of Levy's possible worlds model of general storage [29], with Levy's syntactic Kripke worlds replaced by ***recursively defined univalent semantic worlds***, as can be seen from Computation 3.20 below.

► **Computation 3.20** (Description of adjoints and monad). For the sake of concreteness, we compute the action of the left and right adjoints on objects as follows:

$$\mathsf{F} A\, U = \mathsf{L} \bigcirc \textstyle\sum_{H : \widetilde{\mathcal{H}_{\mathcal{S}}}} \sum_{f : \hom_{\mathcal{C}_{\mathcal{S}}}(U, \pi_{\mathcal{H}} H)} A(\pi_{\mathcal{H}} H)$$
$$\mathsf{U} X\, U = \textstyle\prod_{H : \widetilde{\mathcal{H}_{\mathcal{S}}}} \prod_{f : \hom_{\mathcal{C}_{\mathcal{S}}}(U, \pi_{\mathcal{H}} H)} \mathsf{R}(X(\pi_{\mathcal{H}} H))$$

Composing the above, we describe the action of the monad $\mathsf{IO} = \mathsf{U} \circ \mathsf{F}$ on objects:

$$\mathsf{IO}\, A\, U = \textstyle\prod_{H : \widetilde{\mathcal{H}_{\mathcal{S}}}} \prod_{f : \hom_{\mathcal{C}_{\mathcal{S}}}(U, \pi_{\mathcal{H}} H)} \mathsf{RL} \bigcirc \sum_{H' : \widetilde{\mathcal{H}_{\mathcal{S}}}} \sum_{f : \hom_{\mathcal{C}_{\mathcal{S}}}(\pi_{\mathcal{H}} H, \pi_{\mathcal{H}} H')} A(\pi_{\mathcal{H}} H')$$

► **Lemma 3.21.** *Under the assumptions of Lemma 3.8, each* $\mathsf{U}X$ *is a guarded domain.*

$$\begin{tikzcd}
\underline{\mathcal{N}_{\mathcal{S}}} \arrow[r, bend right, "\mathsf{R}"'] \arrow[rrr, bend right=60, "\mathsf{U}"'] \arrow[r, phantom, "\perp"] & \underline{\bar{\mathcal{P}}_{\mathcal{S}}} \arrow[l, bend right, "\mathsf{L}"'] \arrow[r, bend right, "\underline{\Delta}_{\bar{\pi}_{\mathcal{H}}}"'] \arrow[r, phantom, "\perp"] & \underline{\mathcal{Q}_{\mathcal{S}}} \arrow[l, bend right, "\exists_{\bar{\pi}_{\mathcal{H}}}"'] \arrow[r, bend right, "\underline{\forall}_{\pi_{\mathcal{H}}}"'] \arrow[r, phantom, "\perp"] & \underline{\mathcal{P}_{\mathcal{S}}} \arrow[l, bend right, "\underline{\Delta}_{\pi_{\mathcal{H}}}"'] \arrow[lll, bend right=60, "\mathsf{F}"']
\end{tikzcd}$$

Figure 4 A diagram of $\hat{\mathcal{P}}_{\mathcal{S}}$-enriched adjunctions, together comprising a call-by-push-value adjunction $\mathsf{F} \dashv \mathsf{U} : \underline{\mathcal{N}_{\mathcal{S}}} \to \underline{\mathcal{P}_{\mathcal{S}}}$ resolving an enriched (and thus strong) monad $\mathsf{IO} = \mathsf{U} \circ \mathsf{F}$ on $\underline{\mathcal{P}_{\mathcal{S}}}$.

3.2.3 The model of univalent reference types

We have now defined all the basic elements of our model. All that remains is to define operations corresponding to get, set, *etc.* and to check that they satisfy the equational theory.

3.2.3.1 Fixed-points and store operations

Let $\mathcal{S}$ be a small, set-reflective, guarded subuniverse of a guarded Martin-Löf universe $\mathcal{U}$ containing Inj. Abstract steps are encoded in terms of a global element $\mathsf{step} : \mathsf{IO}\,\mathbf{1}$, defined using the guarded domain structure of $\mathsf{IO}\,\mathbf{1}$ as $\mathsf{step} :\equiv \vartheta_{\mathsf{IO}\,\mathbf{1}}(\mathsf{next}\,(\mathsf{ret}\,*))$. Using $\mathsf{fix}_{\blacktriangleright}$, we can define a monadic fixed point combinator satisfying the equation $\mathsf{rec}\,h\,a = \mathsf{step}; h\,(\mathsf{rec}\,h)\,a$.

We now additionally assume that $\mathcal{S}$ is Σ-closed in order to define the store operations (setting, getting, and allocating) in the resulting call-by-push-value model.

▶ **Construction 3.22** (The getter)**.** For any $A \in \mathcal{P}_{\mathcal{S}}$, we can interpret the getter as a natural transformation $\mathsf{get} : \mathsf{IORef}\,A \to \mathsf{IO}\,A$ in $\mathcal{P}_{\mathcal{S}}$. First we introduce all our arguments:

$$\begin{array}{l}
\mathsf{get} : \mathsf{IORef}\,A \to \mathsf{IO}\,A \\
\mathsf{get}_U(\ell : |U|, \phi : \blacktriangleright[X \leftarrow \partial_U \ell].\lceil X \rceil_U = \lceil A \rceil_U)\,H\,f :\equiv \boxed{\,?: \mathsf{F}A\,(\pi_{\mathcal{H}} H)\,}
\end{array}$$

We proceed by based path induction on the singleton $(\partial_U, \partial_f : \partial_{\pi_{\mathcal{H}} H} \circ |f| = \partial_U)$, setting $U :\equiv (|U|, \partial_{\pi_{\mathcal{H}} H} \circ |f|)$ and $f :\equiv (|f|, \mathsf{refl})$, and then we use the guarded domain structure of the goal to extract the relevant component from the heap, coercing it along the witness that it has the correct type, and leaving the heap untouched.

$$\begin{array}{l}
\mathsf{get}_U(\ell : |U|, \phi : \blacktriangleright[X \leftarrow \partial_U \ell].\lceil X \rceil_U = \lceil A \rceil_U)\,H\,(f \equiv (|f|, \mathsf{refl})) :\equiv \\
\quad \vartheta_{\mathsf{F}A\,(\pi_{\mathcal{H}} H)} \mathsf{next}[X \leftarrow \partial_U \ell, \psi \leftarrow \phi, x \leftarrow H \mathbin{@} |f|\,\ell]. \\
\quad \eta^{\bigcirc}(H, 1_{\pi_{\mathcal{H}} H}, \psi_* x)
\end{array}$$

▶ **Construction 3.23** (The setter)**.** For each $A \in \mathcal{P}_{\mathcal{S}}$, we define the setter as a natural transformation $\mathsf{IORef}\,A \times A \to \mathsf{IO}\,\mathbf{1}$ in $\mathcal{P}_{\mathcal{S}}$.

$$\begin{array}{l}
\mathsf{set} : \mathsf{IORef}\,A \times A \to \mathsf{IO}\,\mathbf{1} \\
\mathsf{set}_U((\ell, \phi), a)\,H\,(f \equiv (|f|, \mathsf{refl})) :\equiv \\
\quad \mathbf{let}\ H_{a/\ell} :\equiv H[|f|\,\ell \hookrightarrow \mathsf{next}[X \leftarrow \partial_U \ell, \psi \leftarrow \phi].\psi_*^{-1}(Af\,a)]\ \mathbf{in}
\end{array}$$

$\mathsf{now}\,(\eta^{\bigcirc}(H_{a/\ell}, 1_{\pi_{\mathcal{H}}H}, *))$

▶ **Construction 3.24** (The allocator). For each $A \in \mathcal{P}_{\mathcal{S}}$, we define the allocator as a natural transformation $A \to \mathsf{IO}\,(\mathsf{IORef}\,A)$ in $\mathcal{P}_{\mathcal{S}}$.

$$\begin{aligned}
&\mathsf{alloc} : A \to \mathsf{IO}\,(\mathsf{IORef}\,A)\\
&\mathsf{alloc}_U\,a\,H\,(f \equiv (|f|, \mathsf{refl})) :\equiv\\
&\quad \mathbf{let}\ H_a :\equiv \begin{cases} \mathsf{inl}\,\ell \hookrightarrow \mathsf{next}[X \leftarrow \partial_{\pi_{\mathcal{H}}H}\ell, x \leftarrow H \mathbin{@} \ell].\,X\,\mathsf{inl}\,x \\ \mathsf{inr}\,* \hookrightarrow \mathsf{next}\,(A\,\mathsf{inr}\,a) \end{cases} \quad \mathbf{in}\\
&\quad \mathsf{now}\,(\eta^{\bigcirc}(H_a, \mathsf{inl}, (\mathsf{inr}\,*, \mathsf{next}\,\mathsf{refl})))
\end{aligned}$$

Above, we have defined a new heap H_a whose underlying finite set of locations is the coproduct $|\pi_{\mathcal{H}}H| + \mathbf{1}$, filling the new location with a and return the pointer to this location.

3.2.3.2 The main theorem

We now come to the main result of this paper, which obtains a model of univalent reference types from a suitably structured small set-reflective subuniverse.

▶ **Theorem 3.25.** *Let $\mathcal{S}$ be a small, Σ-closed, set-reflective, guarded subuniverse of a guarded Martin-Löf universe $\mathcal{U}$ containing* Inj *such that $\mathcal{S}$ is additionally closed under the type of natural numbers. Then there is a model of the monadic language from Section 2 satisfying the equational theory of univalent reference types (Figures 1 and 2), in which:*

1. *contexts, types, and terms are interpreted in the category $\mathcal{P}_{\mathcal{S}} = \mathsf{Fun}(\mathcal{C}_{\mathcal{S}}, \mathsf{Set}_{\mathcal{S}})$;*
2. *the reference type connective is interpreted as in Construction 3.13;*
3. *the computational monad is given by $\mathsf{IO} = \mathsf{U} \circ \mathsf{F}$ as defined in Figure 4;*
4. *general recursion and the store operations are interpreted as in Section 3.2.3.1.*

Proof. We note that $\mathcal{P}_{\mathcal{S}} = \mathsf{Fun}(\mathcal{C}_{\mathcal{S}}, \mathsf{Set}_{\mathcal{S}})$ is locally cartesian closed in spite of the fact that $\mathcal{C}_{\mathcal{S}}$ is as large as $\mathsf{Set}_{\mathcal{S}}$ is: local cartesian closure nonetheless follows because $\mathsf{Set}_{\mathcal{S}}$ is reflective in $\mathcal{U}$ and $\mathcal{C}_{\mathcal{S}}$ is $\mathcal{U}$-small. Everything except the two laws of *univalent* reference types (Figure 2) follows in the same way as in the non-univalent model given by Sterling *et al.* [43]. The ALLOCATION PERMUTATION law holds under the interpretations given because the two heaps resulting from allocations in different orders are identified under univalence. The REPRESENTATION INDEPENDENCE law holds for similar reasons, considering the effect of *transporting* along an identification between equivalent heaps on the getter and the setter. ◀

4 Models of guarded HoTT with impredicative universes

Our main result (Theorem 3.25) is contingent on there existing a model of guarded homotopy type theory in which there can be found a suitably small, Σ-closed, set-reflective, guarded subuniverse of a guarded Martin-Löf universe containing Inj. It is by no means obvious that such a model exists, but in this section we will provide some preliminary evidence.

1. Sterling, Gratzer, and Birkedal [43] have constructed models of ***impredicative guarded dependent type theory*** (**iGDTT**), a non-univalent version of our metalanguage.

2. Awodey [11] has constructed a model of impredicative homotopy type theory in ***cubical assemblies***, *i.e.* internal cubical sets in the category of assemblies. Uemura [45] subsequently described a variant of this model in the style of Orton and Pitts [34].
3. Birkedal *et al.* [15, 14] have constructed an Orton–Pitts model of ***guarded cubical type theory*** in presheaves on the product of a cube category with the ordinal ω. This model was revisited in the context of multi-modal type theory by Aagaard *et al.* [1].

The methods of the papers above are essentially modular, and are furthermore not particularly sensitive to the choice of cube category or ordinal, so long as these can be defined in assemblies without resorting to quotients.

▶ **Conjecture 4.1** (Soundness). *There is a non-trivial model of guarded homotopy type theory in guarded cubical assemblies in which there is a small, set-reflective, guarded Martin-Löf subuniverse $\mathcal{S} \subseteq \mathcal{U}$ of a guarded Martin-Löf universe $\mathcal{U}$ containing* Inj.

5 Conclusions and future work

We have demonstrated the impact of a univalent metalanguage on the denotational semantics of higher-order store, extending the guarded global allocation model of Sterling *et al.* [43] with new program equivalences: invariance under permutation and representation independence in the heap. We believe that we have only scratched the surface of the potential for univalent denotational semantics in general, and univalent reference types in particular; we describe a few potential areas for further development beyond substantiating Conjecture 4.1.

1. Sterling *et al.* [43] have given non-univalent denotational semantics of *polymorphic* λ-calculus with recursive types and general reference types. It is within reach to adapt this model to the univalent setting, obtaining even more free theorems than before. In particular, many data abstraction theorems for existential packages that typically hold only up to observational equivalence are expected to hold on the nose.
2. Our case study, an equation between two object-oriented counters, involves invariance of the heap under *isomorphisms* between data representations — whereas parametricity is often employed in cases of correspondences that are not isomorphisms. Angiuli *et al.* [7] have shown that many such applications of parametricity are nonetheless subsumed by univalence in the presence of quotient types, and thus many more observational equivalences can be replaced with honest equations in univalent denotational semantics. We are eager to put the wisdom of *op. cit.* into practice in the context of imperative and object-oriented programming by incorporating quotients into our theory and model.
3. Although our theory validates many more desirable equations than the global store theory of Sterling *et al.* [43], we do not come close to modeling full local store: for example, two programs that allocate different numbers of cells cannot be equal. We hope that it will be possible to adapt the methods of Kammar *et al.* [27] to the guarded, univalent, and impredicative setting in order to develop even more abstract models of mutable state.
4. Our language does not allow for references to be directly compared (*nominal references*) and no such equality testing function exists in our model. Prior work [44, 33] has given models of such references using the theory of nominal sets [23, 37]. We hope that these methods may be adapted to our model in order to support nominal univalent references.

References

1 Frederik Lerbjerg Aagaard, Magnus Baunsgaard Kristensen, Daniel Gratzer, and Lars Birkedal. Unifying cubical and multimodal type theory. Unpublished manuscript, 2022. doi:10.48550/ARXIV.2203.13000.

2 S. Abramsky, K. Honda, and G. McCusker. A fully abstract game semantics for general references. In *Proceedings of the 13th Annual IEEE Symposium on Logic in Computer Science*, pages 334–344, USA, 1998. IEEE Computer Society. doi:10.1109/LICS.1998.705669.

3 Amal Jamil Ahmed. *Semantics of Types for Mutable State*. PhD thesis, Princeton University, 2004. URL: http://www.ccs.neu.edu/home/amal/ahmedthesis.pdf.

4 Benedikt Ahrens, Dan Frumin, Marco Maggesi, Niccolò Veltri, and Niels van der Weide. Bicategories in univalent foundations. *Mathematical Structures in Computer Science*, 31(10):1232–1269, 2021. doi:10.1017/S0960129522000032.

5 Benedikt Ahrens and Peter LeFanu Lumsdaine. Displayed categories. *Logical Methods in Computer Science*, 15, March 2019. URL: https://lmcs.episciences.org/5252, arXiv:1705.04296, doi:10.23638/LMCS-15(1:20)2019.

6 Pierre America and Jan J. M. M. Rutten. Solving reflexive domain equations in a category of complete metric spaces. In *Proceedings of the 3rd Workshop on Mathematical Foundations of Programming Language Semantics*, pages 254–288, Berlin, Heidelberg, 1987. Springer-Verlag.

7 Carlo Angiuli, Evan Cavallo, Anders Mörtberg, and Max Zeuner. Internalizing representation independence with univalence. *Proceedings of the ACM on Programming Languages*, 5(POPL):1–30, January 2021. doi:10.1145/3434293.

8 Andrew W. Appel and David McAllester. An indexed model of recursive types for foundational proof-carrying code. *ACM Transactions on Programming Languages and Systems*, 23(5):657–683, September 2001. doi:10.1145/504709.504712.

9 Andrew W. Appel, Paul-André Melliès, Christopher D. Richards, and Jérôme Vouillon. 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*, pages 109–122, Nice, France, 2007. Association for Computing Machinery.

10 A. Arnold and M. Nivat. Metric interpretations of infinite trees and semantics of non deterministic recursive programs. *Theoretical Computer Science*, 11(2):181–205, 1980. doi:10.1016/0304-3975(80)90045-6.

11 Steve Awodey. Impredicative encodings in HoTT (or: Towards a realizability ∞-topos). Slides from a talk given the *Big Proof* meeting, Isaac Newton Institute, Cambridge. URL: https://www.andrew.cmu.edu/user/awodey/talks/BigProofs.pdf.

12 Steve Awodey. On Hofmann–Streicher universes. Unpublished manuscript, 2022. doi:10.48550/ARXIV.2205.10917.

13 Steve Awodey, Jonas Frey, and Sam Speight. Impredicative encodings of (higher) inductive types. In *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science*, pages 76–85, Oxford, United Kingdom, 2018. Association for Computing Machinery. doi:10.1145/3209108.3209130.

14 Lars Birkedal, Aleš Bizjak, Ranald Clouston, Hans Bugge Grathwohl, Bas Spitters, and Andrea Vezzosi. Guarded Cubical Type Theory: Path Equality for Guarded Recursion. In Jean-Marc Talbot and Laurent Regnier, editors, *25th EACSL Annual Conference on Computer Science Logic (CSL 2016)*, volume 62 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 23:1–23:17, Dagstuhl, Germany, 2016. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/LIPIcs.CSL.2016.23.

15 Lars Birkedal, Aleš Bizjak, Ranald Clouston, Hans Bugge Grathwohl, Bas Spitters, and Andrea Vezzosi. Guarded cubical type theory. *Journal of Automated Reasoning*, 63(2):211–253, 2019. doi:10.1007/s10817-018-9471-7.

16 Lars Birkedal and Rasmus Ejlers Møgelberg. 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, pages 213–222, Washington, DC, USA, 2013. IEEE Computer Society. doi:10.1109/LICS.2013.27.

17 Lars Birkedal, Rasmus Ejlers Møgelberg, Jan Schwinghammer, and Kristian Støvring. 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*, pages 55–64, Washington, DC, USA, 2011. IEEE Computer Society. arXiv:1208.3596, doi:10.1109/LICS.2011.16.

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

19 Aleš Bizjak, Hans Bugge Grathwohl, Ranald Clouston, Rasmus Ejlers Møgelberg, and Lars Birkedal. Guarded dependent type theory with coinductive types. In Bart Jacobs and Christof Löding, editors, *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*, pages 20–35, Berlin, Heidelberg, 2016. Springer Berlin Heidelberg. arXiv:1601.01586, doi:10.1007/978-3-662-49630-5_2.

20 Aleš Bizjak and Rasmus Ejlers Møgelberg. Denotational semantics for guarded dependent type theory. *Mathematical Structures in Computer Science*, 30(4):342–378, 2020. doi:10.1017/S0960129520000080.

21 Franck Breugel and Jeroen Warmerdam. Solving domain equations in a category of compact metric spaces. Technical report, NLD, 1994.

22 Titouan Carette, Louis Lemonnier, and Vladimir Zamdzhiev. Central submonads and notions of computation: Soundness, completeness and internal languages, 2023. To appear in LICS'23. arXiv:2207.09190.

23 Murdoch J. Gabbay and Andrew M. Pitts. A new approach to abstract syntax with variable binding. *Formal Aspects of Computing*, 13(3):341–363, July 2002. doi:10.1007/s001650200016.

24 Martin Hofmann and Thomas Streicher. Lifting Grothendieck universes. Unpublished note, 1997. URL: https://www2.mathematik.tu-darmstadt.de/~streicher/NOTES/lift.pdf.

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

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

27 Ohad Kammar, Paul B. Levy, Sean K. Moss, and Sam Staton. A monad for full ground reference cells. In *Proceedings of the 32nd Annual ACM/IEEE Symposium on Logic in Computer Science*, Reykjavik, Iceland, June 2017. IEEE Press. doi:10.1109/LICS.2017.8005109.

28 Paul Blain Levy. Possible world semantics for general storage in call-by-value. pages 232–246, September 2002. doi:10.1007/3-540-45793-3_16.

29 Paul Blain Levy. Adjunction models for call-by-push-value with stacks. *Electronic Notes in Theoretical Computer Science*, 69:248–271, 2003. CTCS'02, Category Theory and Computer Science. doi:10.1016/S1571-0661(04)80568-1.

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

31 Rasmus Ejlers Møgelberg and Marco Paviotti. Denotational semantics of recursive types in synthetic guarded domain theory. In *Proceedings of the 31st Annual ACM/IEEE Symposium on Logic in Computer Science*, pages 317–326, New York, NY, USA, 2016. Association for Computing Machinery. doi:10.1145/2933575.2934516.

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

33 Andrzej Murawski and Nikos Tzevelekos. *Foundations and Trends in Programming Languages*, 2(4):191–269, 2016. doi:10.1561/2500000017.

34 Ian Orton and Andrew M. Pitts. Axioms for modelling cubical type theory in a topos. In Jean-Marc Talbot and Laurent Regnier, editors, *25th EACSL Annual Conference on Computer Science Logic (CSL 2016)*, volume 62 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 24:1–24:19, Dagstuhl, Germany, 2016. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/LIPIcs.CSL.2016.24.

35 Marco Paviotti. *Denotational semantics in Synthetic Guarded Domain Theory*. PhD thesis, IT-Universitetet i København, Denmark, 2016.

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

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

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

39 John C. Reynolds. *The Essence of Algol*, pages 67–88. Birkhäuser Boston, Boston, MA, 1997. doi:10.1007/978-1-4612-4118-8_4.

40 Egbert Rijke. Introduction to homotopy type theory. To appear, Cambridge University Press, 2022. doi:10.48550/ARXIV.2212.11082.

41 Egbert Rijke, Michael Shulman, and Bas Spitters. Modalities in homotopy type theory. *Logical Methods in Computer Science*, Volume 16, Issue 1, January 2020. URL: https://lmcs.episciences.org/6015, arXiv:1706.07526, doi:10.23638/LMCS-16(1:2)2020.

42 Sam Staton. Completeness for algebraic theories of local state. In Luke Ong, editor, *Foundations of Software Science and Computational Structures*, pages 48–63, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

43 Jonathan Sterling, Daniel Gratzer, and Lars Birkedal. Denotational semantics of general store and polymorphism. Unpublished manuscript, July 2022. doi:10.48550/arXiv.2210.02169.

44 Nikos Tzevelekos. Full abstraction for nominal general references. *Logical Methods in Computer Science*, Volume 5, Issue 3, 09 2009. doi:10.2168/LMCS-5(3:8)2009.

45 Taichi Uemura. Cubical Assemblies, a Univalent and Impredicative Universe and a Failure of Propositional Resizing. In Peter Dybjer, José Espírito Santo, and Luís Pinto, editors, *24th International Conference on Types for Proofs and Programs (TYPES 2018)*, volume 130 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 7:1–7:20, Dagstuhl, Germany, 2019. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

46 The Univalent Foundations Program. *Homotopy Type Theory: Univalent Foundations of Mathematics*. https://homotopytypetheory.org/book, Institute for Advanced Study, 2013.

47 Philip Wadler. Theorems for free! In *Proceedings of the Fourth International Conference on Functional Programming Languages and Computer Architecture*, FPCA '89, pages 347–359, Imperial College, London, United Kingdom, 1989. Association for Computing Machinery. doi:10.1145/99370.99404.

A Appendix

▶ **Definition 3.11.** *For a universe $\mathcal{S}$, we define the category $\mathcal{C}_{\mathcal{S}}$ of* ***worlds*** *simultaneously with its category of $\mathsf{Set}_{\mathcal{S}}$-valued co-presheaves on $\mathcal{C}_{\mathcal{S}}$ to be the unique solution to the guarded recursive domain equation $\mathcal{C}_{\mathcal{S}} = \mathsf{IBag}_{\blacktriangleright|\mathsf{Fun}(\mathcal{C}_{\mathcal{S}},\mathsf{Set}_{\mathcal{S}})|}$.*

Construction. The system of equations above is solved internally [16] by Löb induction in any guarded Martin-Löf universe $\mathcal{S}^+$ containing $\mathcal{S}$.

$$E :\equiv \mathsf{fix}_{\blacktriangleright}(\lambda R : \blacktriangleright\mathcal{S}^+.|\mathsf{Fun}(\mathsf{IBag}_{\vartheta_{\mathcal{S}^+} R}, \mathsf{Set}_{\mathcal{S}})|) \qquad \mathcal{C}_{\mathcal{S}} :\equiv \mathsf{IBag}_{\blacktriangleright E}$$

Of course, E is the 1-type of objects of the functor category $\mathsf{Fun}(\mathcal{C}_{\mathcal{S}}, \mathsf{Set}_{\mathcal{S}})$. ◀

Let $\mathcal{S}$ be a small, set-reflective, guarded subuniverse of a guarded Martin-Löf universe $\mathcal{U}$.

▶ **Lemma A.1.** *For any $u : \mathsf{IO}\,A$, we have $\mathsf{step}; u = \vartheta_{\mathsf{IO}\,A}(\mathsf{next}\,u)$.*

Proof. This can be seen by unfolding the definition of the $\blacktriangleright$-algebra structure pointwise in the model. ◀

▶ **Construction A.2** (Guarded fixed point combinator)**.** For any $A, B \in \mathcal{P}_{\mathcal{S}}$, we can define a monadic fixed point combinator.

$$\mathsf{rec} : ((A \to \mathsf{IO}\,B) \to A \to \mathsf{IO}\,B) \to A \to \mathsf{IO}\,B$$
$$\mathsf{rec}\,h :\equiv \mathsf{fix}_{\blacktriangleright}\,(\lambda f.\,\lambda x.\,\vartheta_{\mathsf{IO}\,B}(\mathsf{next}[g \leftarrow f].h\,g\,x))$$

▶ **Lemma A.3.** *We have $\mathsf{rec}\,h\,a = \mathsf{step}; h\,(\mathsf{rec}\,h)\,a$.*

Proof. We compute as follows:

$$\begin{aligned}
&\mathsf{rec}\,h\,a\\
&\quad\text{by unfolding definitions}\\
&\quad\equiv \mathsf{fix}_{\blacktriangleright}\,(\lambda f.\,\lambda x.\,\vartheta_{\mathsf{IO}\,B}(\mathsf{next}[g \leftarrow f].h\,g\,x))\,a\\
&\quad\text{by } \mathsf{fix}_{\blacktriangleright} \text{ computation rule}\\
&\quad\equiv \vartheta_{\mathsf{IO}\,B}(\mathsf{next}[g \leftarrow \mathsf{next}\,(\mathsf{rec}\,h)].h\,g\,a)\\
&\quad\text{by rules of delayed substitutions}\\
&\quad\equiv \vartheta_{\mathsf{IO}\,B}(\mathsf{next}\,(h\,(\mathsf{rec}\,h)\,a))\\
&\quad\text{by Lemma A.1}\\
&\quad= \mathsf{step}; h\,(\mathsf{rec}\,h)\,a
\end{aligned}$$

We are done. ◀

Now suppose in addition that $\mathcal{S}$ is Σ-closed.

▶ **Construction A.4** (Detailed construction of the getter)**.** For any $A \in \mathcal{P}_{\mathcal{S}}$, we can interpret the getter as a natural transformation $\mathsf{get} : \mathsf{IORef}\,A \to \mathsf{IO}\,A$ in $\mathcal{P}_{\mathcal{S}}$. Because the definition is a little subtle, we will do it step-by-step.

$$\mathsf{get} : \mathsf{IORef}\,A \to \mathsf{IO}\,A$$
$$\mathsf{get}_U(\ell : |U|, \phi : \blacktriangleright[X \leftarrow \partial_U \ell].\lceil X\rceil_U = \lceil A\rceil_U)\,H\,f :\equiv \boxed{?:\mathsf{F}A\,(\pi_{\mathcal{H}} H)}$$

We proceed by based path induction on the singleton $(\partial_U, \partial_f : \partial_{\pi_{\mathcal{H}} H} \circ |f| = \partial_U)$, setting $U :\equiv (|U|, \partial_{\pi_{\mathcal{H}} H} \circ |f|)$ and $f :\equiv (|f|, \mathsf{refl})$:

$$\mathsf{get}_U(\ell : |U|, \phi : \blacktriangleright[X \leftarrow \partial_U \ell].\lceil X\rceil_U = \lceil A\rceil_U)\,V\,(f \equiv (|f|, \mathsf{refl})) :\equiv \boxed{?:\mathsf{F}A\,(\pi_{\mathcal{H}} H)}$$

Next, we use the guarded domain structure of the goal:

$$\mathsf{get}_U(\ell : |U|, \phi : \blacktriangleright[X \leftarrow \partial_U \ell].\lceil X\rceil_U = \lceil A\rceil_U)\, H\,(f \equiv (|f|, \mathsf{refl})) :\equiv \\ \vartheta_{\mathsf{F}A\,(\pi_{\mathcal{H}} H)}\ ?: \blacktriangleright \mathsf{F}A\,(\pi_{\mathcal{H}} H)$$

Using the introduction rule for the later modality, we may unwrap the delayed identification ϕ to assume $\psi : \lceil X\rceil_U = \lceil A\rceil_U$, as well as the delayed element $H @ |f|\,\ell$ to assume $x : X(\pi_{\mathcal{H}} H)$:

$$\mathsf{get}_U(\ell : |U|, \phi : \blacktriangleright[X \leftarrow \partial_U \ell].\lceil X\rceil_U = \lceil A\rceil_U)\, H\,(f \equiv (|f|, \mathsf{refl})) :\equiv \\ \vartheta_{\mathsf{F}A\,(\pi_{\mathcal{H}} H)} \mathsf{next}[X \leftarrow \partial_U \ell, \psi \leftarrow \phi, x \leftarrow H @ |f|\,\ell].\ ?: \mathsf{F}A\,(\pi_{\mathcal{H}} H)$$

Applying the unit of the reflection $\bigcirc : \mathcal{U} \to \mathsf{Set}_{\mathcal{S}}$ and splitting the resulting goal, we have three holes:

$$\mathsf{get}_U(\ell : |U|, \phi : \blacktriangleright[X \leftarrow \partial_U \ell].\lceil X\rceil_U = \lceil A\rceil_U)\, H\,(f \equiv (|f|, \mathsf{refl})) :\equiv \\ \vartheta_{\mathsf{F}A\,(\pi_{\mathcal{H}} H)} \mathsf{next}[X \leftarrow \partial_U \ell, \psi \leftarrow \phi, x \leftarrow H @ |f|\,\ell]. \\ \eta^{\bigcirc}(\ ?_0 : \widetilde{\mathcal{H}_{\mathcal{S}}}\ ,\ ?_1 : \hom_{\mathcal{C}_{\mathcal{S}}}(\pi_{\mathcal{H}} H, \pi_{\mathcal{H}} ?_0)\ ,\ ?_2 : A(\pi_{\mathcal{H}} ?_0)\ )$$

A read-operation does not change the heap; therefore, we fill in the first hole with the existing heap H and the second hole with the identity map.

$$\mathsf{get}_U(\ell : |U|, \phi : \blacktriangleright[X \leftarrow \partial_U \ell].\lceil X\rceil_U = \lceil A\rceil_U)\, H\,(f \equiv (|f|, \mathsf{refl})) :\equiv \\ \vartheta_{\mathsf{F}A\,(\pi_{\mathcal{H}} H)} \mathsf{next}[X \leftarrow \partial_U \ell, \psi \leftarrow \phi, x \leftarrow H @ |f|\,\ell]. \\ \eta^{\bigcirc}(H, 1_{\pi_{\mathcal{H}} H},\ ?: A(\pi_{\mathcal{H}} H)\ )$$

Recall that we have an identification $\psi : \lceil X\rceil_U = \lceil A\rceil_U$ in the type $\lfloor \mathsf{Set}_{\mathcal{S}}\rfloor_U$ of copresheaves on $U/\mathcal{C}_{\mathcal{S}}$; transporting by this identification in the family $Z : \lfloor \mathsf{Set}_{\mathcal{S}}\rfloor_U \vdash Z\,(\pi_{\mathcal{H}} H)\, f : \mathsf{Set}_{\mathcal{S}}$, we have a mapping from $\lceil X\rceil_U\,(\pi_{\mathcal{H}} H)\, f \equiv X(\pi_{\mathcal{H}} H)$ to $\lceil A\rceil_U\,(\pi_{\mathcal{H}} H)\, f \equiv A(\pi_{\mathcal{H}} H)$, which we use to fill the final hole:

$$\mathsf{get}_U(\ell : |U|, \phi : \blacktriangleright[X \leftarrow \partial_U \ell].\lceil X\rceil_U = \lceil A\rceil_U)\, H\,(f \equiv (|f|, \mathsf{refl})) :\equiv \\ \vartheta_{\mathsf{F}A\,(\pi_{\mathcal{H}} H)} \mathsf{next}[X \leftarrow \partial_U \ell, \psi \leftarrow \phi, x \leftarrow H @ |f|\,\ell]. \\ \eta^{\bigcirc}(H, 1_{\pi_{\mathcal{H}} H}, \psi_* x)$$

This completes the definition of the getter.

► **Construction A.5** (Detailed construction of the setter)**.** For each $A \in \mathcal{P}_{\mathcal{S}}$, we define the setter as a natural transformation $\mathsf{IORef}\,A \times A \to \mathsf{IO}\,\mathbf{1}$ in $\mathcal{P}_{\mathcal{S}}$.

$$\mathsf{set} : \mathsf{IORef}\,A \times A \to \mathsf{IO}\,\mathbf{1} \\ \mathsf{set}_U((\ell, \phi), a)\, H\,(f \equiv (|f|, \mathsf{refl})) :\equiv\ ?: \mathsf{F}\mathbf{1}\,(\pi_{\mathcal{H}} H)$$

We apply the unit now of the lifting monad, followed by the unit of the reflection $\bigcirc : \mathcal{U} \to \mathsf{Set}_{\mathcal{S}}$, and then split the goal:

$$\mathsf{set} : \mathsf{IORef}\,A \times A \to \mathsf{IO}\,\mathbf{1} \\ \mathsf{set}_U((\ell, \phi), a)\, H\,(f \equiv (|f|, \mathsf{refl})) :\equiv \\ \mathsf{now}\,(\eta^{\bigcirc}(\ ?_0 : \widetilde{\mathcal{H}_{\mathcal{S}}}\ ,\ ?_1 : \hom_{\mathcal{C}_{\mathcal{S}}}(\pi_{\mathcal{H}} H, \pi_{\mathcal{H}} ?_0)\ , *))$$

We want to replace the contents of H at the location $|f|\,\ell$ with the reindexed element $\mathsf{next}(Af\,a) : \blacktriangleright A(\pi_{\mathcal{H}} H)$; for this to make sense, we need to transport along the (delayed) identification $\phi : \blacktriangleright[X \leftarrow \partial_U \ell].\lceil X\rceil_U = \lceil A\rceil_U$. We define the updated heap as follows, noting that $\partial_U \ell \equiv \partial_{\pi_{\mathcal{H}} H} |f|\,\ell$:

$?_0 :\equiv H[|f|\,\ell \hookrightarrow \mathsf{next}[X \leftarrow \partial_U \ell, \psi \leftarrow \phi].\psi_*^{-1}(Af\,a)]$

The updated heap can be so-defined because its set of locations is finite, and thus has decidable equality. Because the updated heap has the same underlying configuration, we can fill our remaining hole $?_1 :\equiv 1_{\pi_{\mathcal{H}} H}$, completing the definition of the setter as follows:

$\mathsf{set} : \mathsf{IORef}\, A \times A \to \mathsf{IO}\,\mathbf{1}$
$\mathsf{set}_U\,((\ell, \phi), a)\,H\,(f \equiv (|f|, \mathsf{refl})) :\equiv$
$\quad \mathbf{let}\ H_{a/\ell} :\equiv H[|f|\,\ell \hookrightarrow \mathsf{next}[X \leftarrow \partial_U \ell, \psi \leftarrow \phi].\psi_*^{-1}(Af\,a)]\ \mathbf{in}$
$\quad \mathsf{now}\,(\eta^{\bigcirc}(H_{a/\ell}, 1_{\pi_{\mathcal{H}} H}, *))$