Towards univalent reference types

The impact of univalence on denotational semantics

Jonathan Sterling
University of Cambridge

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 program equivalences. In particular, even the most simplistic univalent model enjoys many new equations 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

Digital Object Identifier 10.4230/LIPIcs.CSL.2024.19

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 was funded in part by the European Union under the Marie Skłodowska-Curie Actions Postdoctoral Fellowship project TypeSynth: synthetic methods in program verification, and in part by AFOSR under grant FA9550-23-1-0728, New Spaces for Denotational Semantics (Tristan Nguyen, program manager). Views and opinions expressed are however those of the authors only and do not necessarily reflect those of the European Union, the European Commission, nor AFOSR. Neither the European Union nor the granting authority nor AFOSR 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. Finally, we are grateful to the anonymous referees for their comments and suggestions.

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.

© Jonathan Sterling, Daniel Gratzer, and Lars Birkedal;
licensed under Creative Commons License CC-BY 4.0
32nd EACSL Annual Conference on Computer Science Logic (CSL 2024).
Editors: Aniello Murano and Alexandra Silva; Article No. 19; pp. 19:1–19:21
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany
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 $\ell_i : \sigma_i$ can be classified by Moggi’s state monad $\sigma \rightarrow \sigma \times -$ where $\sigma := \prod_i \sigma_i$, 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 $(\ell \leftarrow \text{alloc } \text{"hello"}; \text{ret } 10)$ from the simpler program $\text{ret } 10$. By contrast, models of local store validate equations resembling an idealized garbage collector, in which the heap is only observable through its abstract read/write interface; in a model of local store, we necessarily have $(\ell \leftarrow \text{alloc } \text{"hello"}; \text{ret } 10) = \text{ret } 10$ 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 so a landscape of intermediate theories has emerged in the search for well-behaved models. 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 limitation: 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 a 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 `IORef σ` 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 presented evidence that such a fixed point 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]). This model 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 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].
Towards univalent reference types

Univalent reference types and data abstraction 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:

\[ \text{Counter} \equiv \{ \text{incr} : \text{IO \{\}}; \text{read} : \text{IO \text{Int}} \} \]

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

\[ \text{posCounter} : \text{IO \text{Counter}} \]
\[ \text{posCounter} \equiv \ell \leftarrow \text{alloc} 0; \text{ret} \{ \text{incr} \leftrightarrow i \leftrightarrow \text{get} \ell; \text{set} \ell (i + 1), \text{read} \leftrightarrow \text{get} \ell \} \]

Another implementation might count backwards and then negate the stored value on read using the functorial action map of IO on neg: Int \( \rightarrow \) Int:

\[ \text{negCounter} : \text{IO \text{Counter}} \]
\[ \text{negCounter} \equiv \ell \leftarrow \text{alloc} 0; \text{ret} \{ \text{incr} \leftrightarrow i \leftrightarrow \text{get} \ell; \text{set} \ell (i - 1), \text{read} \leftrightarrow \text{map neg (get} \ell) \} \]

By intuition, the posCounter and 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 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 posCounter \( \simeq \) 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 \( \vdash \text{posCounter} \equiv \text{negCounter} \), as can be seen easily by means of a countermodel: we have \( \text{posCounter} \neq \text{negCounter} \) in both the relational models of op. cit., although it is true that \( \text{posCounter} R_{\text{IO \text{Counter}}} \text{negCounter} \) holds. The observational equivalence posCounter \( \simeq \) 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 \( \vdash \text{posCounter} \equiv \text{negCounter} \), as shown in Theorem 2.1. We substantiate this equational theory by constructing a model (Theorem 3.27) in univalent foundations [46] in which the equation \( \vdash \text{posCounter} \equiv \text{negCounter} \) follows immediately from the univalence principle of the metalanguage. Although it may be possible to validate this equation using non-standard parametric models (or less scrupulously by an extensional collapse), our contribution is to show that it also holds in a “standard” model, provided that this standard model is constructed in a univalent metatheory.

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 several problematic constructs (higher-order store, dynamic allocations, etc.) that have been historically difficult to model in denotational semantics.
Although there are many different ways to present programming languages with side effects, when developing our denotational semantics in Section 3, we will refine the monadic point of view by

2.1 The equational theory of monadic general reference types

Although 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]. Essentially, this is a simply-typed lambda calculus supplemented with a strong monad \( \text{IO} \) and further equipped with a type of references \( \text{IORef} \) 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.

One non-standard aspect of our language bears special attention, namely the nullary side effect \( \text{step} : \text{IO} () \). 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 \( g : \text{IO} x \in e \) we do not have \( \vdash g \equiv [g/f, e'/x] e \) but rather only \( \vdash [g/f, e'/x] e \equiv \text{step} ; [g/f, e'/x] e \). Likewise, our equational theory does not equate \( \ell \leftarrow \text{alloc} e ; \text{get} \ell \equiv \text{step} ; \text{ret} e \). The presence of \( \text{step} \) in our equational theory is forced by

\[ e : \text{IORef} \sigma, e' : \sigma \vdash \text{set} e e' \equiv \text{step} ; \text{set} e e' : \text{IO} \sigma \]
\[ e : \sigma, e' : \sigma \vdash (x \leftarrow \text{alloc} e ; \text{set} x e' ; \text{ret} x) \equiv \text{alloc} e' : \text{IO} (\text{IORef} \sigma) \]
\[ e : \text{IORef} \sigma, e' : \sigma, e'' : \sigma \vdash \text{set} e e' : \text{IO} () \equiv \text{set} e e'' : \text{IO} () \]
\[ e : \text{IORef} \sigma, e' : \text{IORef} \sigma \vdash (x \leftarrow \text{get} e ; y \leftarrow \text{get} e ; \text{ret} \langle x, y \rangle) \equiv (y \leftarrow \text{get} e ; x \leftarrow \text{get} e ; \text{ret} \langle x, y \rangle) : \text{IO} (\sigma \times \tau) \]
\[ e : \text{IORef} \sigma \vdash (x \leftarrow \text{get} e ; \text{set} x e ; \text{ret} x) \equiv \text{get} e : \text{IO} \sigma \]
\[ e : \text{IORef} \sigma, e' : \text{IO} \tau \vdash \text{get} e ; e' \equiv \text{step} ; e' \]

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 \( e \leftarrow e' \). We assume the standard \( \beta/\eta \)-equational theory of function and product types, as well as the monadic laws. We also assume that \( \text{step} \) lies in the center [22] of the monad \( \text{IO} \), i.e., commutes with all monadic operations.

2.1 The equational theory of monadic general reference types

Although 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]. Essentially, this is a simply-typed lambda calculus supplemented with a strong monad \( \text{IO} \) and further equipped with a type of references \( \text{IORef} \) 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.

One non-standard aspect of our language bears special attention, namely the nullary side effect \( \text{step} : \text{IO} () \). 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 \( g : \text{rec} x \in e \), we do not have \( \vdash g \equiv [g/f, e'/x] e \) but rather only \( \vdash [g/f, e'/x] e \equiv \text{step} ; [g/f, e'/x] e \). Likewise, our equational theory does not equate \( \ell \leftarrow \text{alloc} e ; \text{get} \ell \equiv \text{step} ; \text{ret} e \). The presence of \( \text{step} \) in our equational theory is forced by

\[ e : \text{IORef} \sigma, e' : \sigma \vdash \text{set} e e' ; \text{get} e \equiv \text{step} ; \text{set} e e' ; \text{ret} e' : \text{IO} \sigma \]
\[ e : \sigma, e' : \sigma \vdash (x \leftarrow \text{alloc} e ; \text{set} x e' ; \text{ret} x) \equiv \text{alloc} e' : \text{IO} (\text{IORef} \sigma) \]
\[ e : \text{IORef} \sigma, e' : \sigma, e'' : \sigma \vdash \text{set} e e' : \text{IO} () \equiv \text{set} e e'' : \text{IO} () \]
\[ e : \text{IORef} \sigma, e' : \text{IORef} \sigma \vdash (x \leftarrow \text{get} e ; y \leftarrow \text{get} e ; \text{ret} \langle x, y \rangle) \equiv (y \leftarrow \text{get} e ; x \leftarrow \text{get} e ; \text{ret} \langle x, y \rangle) : \text{IO} (\sigma \times \tau) \]
\[ e : \text{IORef} \sigma \vdash (x \leftarrow \text{get} e ; \text{set} x e ; \text{ret} x) \equiv \text{get} e : \text{IO} \sigma \]
\[ e : \text{IORef} \sigma, e' : \text{IO} \tau \vdash \text{get} e ; e' \equiv \text{step} ; e' \]

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 resolution of the computational monad [29].
Towards univalent reference types

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.

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 \( \sigma \) and then only interacting with it by means of its (get, set) methods should be the same as allocating a cell of a different type \( \tau \) and interacting with it by conjugating its (get, 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.

\[ 2.2 \] 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.
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 \( \sigma \) and then only interacting with it by means of its (get, set) methods should be the same as allocating a cell of a different type \( \tau \) and interacting with it by conjugating its (get, 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.

\[ \text{Theorem 2.1.} \] Let Counter, posCounter, and negCounter be as in Section 1.4:

\[
\begin{align*}
\text{Counter} & : \equiv \{ \text{incr} : \text{IO}(); \text{read} : \text{IO} \text{Int} \} \\
\text{posCounter} & : \equiv \ell \leftarrow \text{alloc}(); \text{ret} \{ \text{incr} \leftarrow i \leftarrow \text{get} \ell; \text{set} \ell (i + 1), \text{read} \leftarrow \text{get} \ell \} \\
\text{negCounter} & : \equiv \ell \leftarrow \text{alloc}(); \text{ret} \{ \text{incr} \leftarrow i \leftarrow \text{get} \ell; \text{set} \ell (i - 1), \text{read} \leftarrow \text{map neg} (\text{get} \ell) \}
\end{align*}
\]

We may derive \( \vdash \) posCounter \( \equiv \) negCounter : Counter.

\[ \text{Proof.} \] The function neg : Int \( \rightarrow \) Int sending an integer to its negation is a self-dual automorphism; we therefore calculate from left to right.
\[ \ell \leftarrow \text{alloc } 0; \text{ret } \{ \text{incr } \rightarrow i \leftarrow \text{get } \ell; \text{set } \ell (i + 1), \text{read } \rightarrow \text{get } \ell \} \]

by representation independence

\[ \equiv \ell \leftarrow \text{alloc } (\text{neg } 0); \text{ret } \{ \text{incr } \rightarrow i \leftarrow \text{map } \text{neg } (\text{get } \ell); \text{set } \ell (\text{neg } (i + 1)), \text{read } \rightarrow \text{map } \text{neg } (\text{get } \ell) \} \]

by simplification and neg 0 \equiv 0

\[ \equiv \ell \leftarrow \text{alloc } 0; \text{ret } \{ \text{incr } \rightarrow i \leftarrow \text{get } \ell; \text{set } \ell (\text{neg } (i + 1)), \text{read } \rightarrow \text{map } \text{neg } (\text{get } \ell) \} \]

by neg (neg i + 1) \equiv neg (neg i) + neg 1 \equiv i - 1

\[ \equiv \ell \leftarrow \text{alloc } 0; \text{ret } \{ \text{incr } \rightarrow i \leftarrow \text{get } \ell; \text{set } \ell (i - 1), \text{read } \rightarrow \text{map } \text{neg } (\text{get } \ell) \} \]

Thus we have posCounter \equiv 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 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 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...
Towards univalent reference types

unique) function \( \eta : A \to A_U \) with \( A_U \in U \) such for any type \( C \in U \), the precomposition map \( C^\eta : C^{A_U} \to C^A \) is an equivalence. When a reflection of \( A \) in \( U \) exists (necessarily uniquely), we shall say that \( A \) is reflected in \( U \).

A subuniverse of a universe \( U \) is defined to be a dependent type \( A : U \vdash PA \) such that each \( PA \) is a proposition. We may write \( U_P \) for the universe \( \sum_{A : U} PA \) obtained by restricting \( U \) 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 \( S \subseteq U \) is said to be reflective if every \( A : U \) is reflected in \( S \). A subuniverse of \( U \) is said to be “small” when its comprehension as a type is \( U \)-small.

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

> Theorem 3.1. A small \( \Sigma \)-closed subuniverse \( S \) of a Martin-Löf universe \( U \) 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 \( S \) be a small subuniverse of a Martin-Löf universe \( U \), and let \( \mathcal{X} \) be a \( U \)-small category; we can define the Hofmann–Streicher lifting [24, 12] of \( \text{Set}_S \) 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 \( \text{1Type}_U \) of 1-types in \( U \); thus we depend technically on the account of bicategories in univalent foundations due to Ahrens et al. [4].

> 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 \( S \) be a small subuniverse of a Martin-Löf universe \( U \), and let \( \mathcal{X} \) be a \( U \)-small category. We may define a 2-functor \( [\text{Set}_S] : \mathcal{X} \to \text{1Type}_U \) called the Hofmann–Streicher lifting of \( \text{Set}_S \) as follows:

\[
[\text{Set}_S](U) \equiv [\text{Fun}(U/\mathcal{X}, \text{Set}_S)]
\]

\[
[\text{Set}_S](f : U \to V)(E : U/\mathcal{X} \to \text{Set}_S) \equiv E \circ (f/\mathcal{X})
\]

When \( \mathcal{X} \) is viewed as a 2-category, the 2-cells are given by identifications. Thus the 2-functoriality of \( [\text{Set}_S] \) and all related coherences are defined by path induction.

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.
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.

3.1.3 (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 ⊢ ξ ⊳ Ξ simultaneously with a new type former ⊢ [ξ].A type called the later modality, whose introduction form is written \text{next}[ξ].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 \( f : \uparrow A \rightarrow A \), there is a unique element \text{fix}_\uparrow f : A \) such that \( f (\text{next}(\text{fix}_\uparrow f)) = \text{fix}_\uparrow f \). In particular, this gives unique fixed points for any function \( f : A \rightarrow A \) factoring on the left through \( \text{next} : A \rightarrow \uparrow A \).

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

\[ f : \uparrow A \rightarrow A \quad \frac{\text{fix}_\uparrow f : A}{f : \uparrow A \rightarrow A} \]

\[ f : \uparrow A \rightarrow A \quad \frac{\text{fix}_\uparrow f \equiv f (\text{next}(\text{fix}_\uparrow f))}{f : \uparrow A \rightarrow A} \]

\( \triangleright \) Construction 3.4 (Restricting co-presheaves). Let \( S \) be a small subuniverse of a Martin-Löf universe \( \mathcal{U} \), and let \( X \) be a \( \mathcal{U} \)-small category. Every co-presheaf \( E : X \rightarrow \text{Set}_S \) determines a global element \( 1 \rightarrow [\text{Set}_S] \) of the Hofmann–Streicher universe; in particular, we may define \( [E]_U : [\text{Set}_S]U \) natural in \( U \in X \) by setting \( [E]_U (f : U \rightarrow V) : \equiv EV \).

We define \( \text{fix}_\uparrow A \) equipped with a mapping \( \vartheta_A : \uparrow A \rightarrow A \) that \( \uparrow \) -algebra, i.e. a function \( \vartheta_A : \uparrow A \rightarrow A \).

\( \triangleright \) Lemma 3.6. If \( S \) is a guarded universe closed under binary coproducts, then the forgetful functor \( R : \text{0Dom}_S \rightarrow \text{Set}_S \) has a left adjoint \( L : \text{Set}_S \rightarrow \text{0Dom}_S \).

\( \triangleright \) Notation 3.7. We will write \( \text{now} : A \rightarrow RLA \) for the unit of the adjunction \( L \dashv R \).
Towards univalent reference types

Lemma 3.8 (Later modality in presheaves). Given a guarded set-reflective small subuniverse $S \subseteq U$ and a $U$-small category $X$, the later modality from $S$ lifts (with all its operations) into $Fun(X, Set_S)$ pointwise, i.e. for any $A \in Fun(X, Set_S)$ we may define $(\triangleright A)U \equiv \triangleright (AU)$.

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).

3.2.1 Worlds as univalent heap configurations

Let $\text{Inj}$ be the category of finite sets and embeddings; by univalence, any two equipollent $\text{Inj}$-heaps are identified. We now define the basic elements of worlds $\text{qua}$ heap configurations.

Definition 3.9 (The displayed category of families). For any 1-type $X$, we define the displayed category $[S ] l f a m_X$ of $\text{Inj}$-families in $X$ over $\text{Inj}$ as follows:
1. over a finite set $I : \text{Inj}$, a displayed object of $lfam_X$ is a function $\partial_I : I \to X$;
2. over a function $f : I \to J$ between finite sets, a displayed morphism from $\partial_I$ to $\partial_J$ 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 $\text{Inj}$-bags in $X$ is defined to be the total category $l b a g_X \equiv \int_{\text{Inj}} l f a m_X$ of the displayed category of finite families in $X$. We will write $U \equiv ([U], \partial_U)$ for an object of $l b a g_X$.

Definition 3.11. For a universe $S$, we define the category $C_S$ of worlds simultaneously with its category of $\text{Sets}_S$-valued co-presheaves on $C_S$ to be the unique solution to the guarded recursive domain equation $C_S = l b a g^{\triangleright}(Fun(C_S, \text{Sets}_S))$.

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

$E \equiv \text{fix}_{\triangleright}(\lambda R : \triangleright S^+ ; Fun(l b a g_{\triangleright S^+ R}, \text{Sets}_S)) \quad C_S \equiv l b a g^{\triangleright} E$

Of course, $E$ is the 1-type of objects of the functor category $Fun(C_S, \text{Sets}_S)$.

We shall require the following technical observation:

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

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

Construction 3.13 (Univalent references). Let $S$ be a small, guarded, $\Sigma$-closed set-reflective subuniverse of a guarded Martin-Löf universe $U$ containing $\text{Inj}$. We define the univalent reference type constructor as a mapping $\text{IORef} : \text{Fun}(C_S, \text{Sets}_S) \to \text{Fun}(C_S, \text{Sets}_S)$:

$\text{IORef} : \text{Fun}(C_S, \text{Sets}_S) \to \text{Fun}(C_S, \text{Sets}_S)$

$\text{IORef} AU \equiv \sum_{E : [U]} \triangleright [X \leftarrow \partial_U E], [X]_U = [A]_U$
Above, we have used the \([-\cdot]\) operator from Construction 3.4. We define the functorial action of \(\text{IORef} A\) on \(f : U \to V\) by path induction on the identification \(\partial_f : \partial_V \circ |f| = \partial_U:\)

\[
\text{IORef} A ([|f|, \text{refl}])(\ell, \phi) \equiv ([|f|, \text{next}[X \leftarrow \partial_V(|f|\ell), \psi \leftarrow \phi], \text{ap}_{\text{Set}_S}|f|\psi)
\]

Proof. That the identification \([X]_U = [A]_U\) is \(\mathcal{S}\)-small follows from Lemma 3.12, using the fact that \(U/\mathcal{C}_S\) 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.\(^5\) 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.

\[\blacktriangleright\text{Construction 3.14 (The heaplet distributor).} \] Let \(\mathcal{S}\) be guarded universe closed under finite products. We may define a distributor \(\mathcal{H}_\mathcal{S} : \mathcal{C}_\mathcal{S}^{\mathcal{S}} \times \mathcal{C}_\mathcal{S} \to \text{Set}_\mathcal{S}\) like so:

\[
\mathcal{H}_\mathcal{S}(U, V) \equiv \prod_{\ell \in U} (\mathcal{H}_\mathcal{S}(\mathcal{S}_{\mathcal{S}})^{\mathcal{S}}(\ell \circ ) \mathcal{H}_\mathcal{S}(\mathcal{S}_{\mathcal{S}})^{\mathcal{S}}(\ell \circ )  V)
\]

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

**Presheaf categories and unenriched adjunctions**

Let \(\mathcal{S}\) be guarded universe closed under finite products. As \(\mathcal{H}_\mathcal{S}\) 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 \(\pi_\mathcal{S}_U : \mathcal{H}_\mathcal{S} \to \mathcal{C}_\mathcal{S}\) extends to functors \(\pi_\mathcal{S}_U : \mathcal{H}_\mathcal{S} \to \mathcal{C}_\mathcal{S}\) and \(\pi_{\mathcal{S}^o} : \mathcal{H}_\mathcal{S}^{\mathcal{S}} \simeq \mathcal{H}_\mathcal{S}^{\mathcal{S}^o} \rightarrow \mathcal{C}_\mathcal{S}^{\mathcal{S}}\). We will use these projections to construct a network of adjunctions between the following presheaf categories:

\[
\begin{align*}
\mathcal{P}_\mathcal{S} & \equiv \text{Fun}(\mathcal{C}_\mathcal{S}, \text{Set}_\mathcal{S}) & \mathcal{N}_\mathcal{S} & \equiv \text{Fun}(\mathcal{C}_\mathcal{S}^{\mathcal{S}^o}, \text{0Dom}_\mathcal{S}) \\
\mathcal{P}_\mathcal{S}^o & \equiv \text{Fun}(\mathcal{C}_\mathcal{S}^{\mathcal{S}}, \text{Set}_\mathcal{S}) & \mathcal{Q}_\mathcal{S} & \equiv \text{Fun}(\mathcal{H}_\mathcal{S}, \text{Set}_\mathcal{S})
\end{align*}
\]

\[\blacktriangleright\text{Exegesis 3.15.} \] \(\mathcal{P}_\mathcal{S}\) 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.

\(^5\) The purpose of strength, as ever, to transform the global Kleisli extension operation of the monad into a \textit{binding-operation} that applies in arbitrary contexts.
Towards univalent reference types

by factoring the adjunction $F \dashv U : \mathcal{N}_S \to \mathcal{P}_S$ 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 $S$ be a small, set-reflective, guarded subuniverse of a guarded Martin-Löf universe $\mathcal{U}$ containing $\text{Inj}$. Then the unenriched base change functors $\Delta_{\pi_S} : \mathcal{P}_S \to \mathcal{Q}_S$ and $\Delta_{\bar{\pi}_S} : \bar{\mathcal{P}}_S \to \bar{\mathcal{Q}}_S$ has left and right adjoints $\exists_{\bar{\pi}_S} \dashv \Delta_{\bar{\pi}_S}$ and $\Delta_{\pi_S} \dashv \forall_{\pi_S}$ respectively.

**Proof.** As $\bar{\mathcal{P}}_S$ is both discrete and $\mathcal{U}$-small, the Kan extensions exist because $\text{Set}_S$ 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 $\circ : \mathcal{U} \to \text{Set}_S$ is the assumed reflection:

$$
\exists_{\bar{\pi}_S} A U = \circ \sum_{H : \bar{\mathcal{P}}_S} \sum_{f : \text{hom}_{\bar{\mathcal{P}}_S}(\bar{\pi}_S H, U)} \text{AH}
$$

$$
\forall_{\pi_S} A U = \prod_{H : \mathcal{P}_S} \prod_{f : \text{hom}_{\mathcal{P}_S}(\pi_S H, U)} \text{AH}
$$

We draw attention to the fact that codomain of $\exists_{\bar{\pi}_S}$ is $\bar{\mathcal{P}}_S$ while the codomain of $\forall_{\pi_S}$ is $\mathcal{P}_S$, hence the appearance of $\text{hom}_{\bar{\mathcal{P}}_S}$ in the former and $\text{hom}_{\mathcal{P}_S}$ in the latter.

**Enrichments and enriched adjunctions**

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

$$
\mathcal{P}_S^\Gamma := \text{Fun}(\Gamma^{\mathcal{P}_S}, \text{Set}_S) \\
\bar{\mathcal{P}}_S^\Gamma := \text{Fun}(\bar{\Gamma}, \text{Set}_S) \\
\mathcal{N}_S^\Gamma := \text{Fun}(\Gamma, \text{0Dom}_S) \\
\bar{\mathcal{Q}}_S^\Gamma := \text{Fun}(\Gamma^{\mathcal{Q}_S}, \text{Set}_S)
$$

Above, we note that $\mathcal{P}_S^\Gamma$ is equivalent to the slice $\mathcal{P}_S/\Gamma$; in the definition of $\mathcal{Q}_S^\Gamma$, the expression $\Gamma^{\mathcal{P}_S} \times _{\mathcal{E}_S} \bar{\mathcal{P}}_S$ refers to the pullback of the span $\{\Gamma^{\mathcal{P}_S} \xrightarrow{\pi_\mathcal{P}_S} \mathcal{E}_S \xrightarrow{\pi_\mathcal{Q}_S} \bar{\mathcal{P}}_S\}$. We have the following base change functors for any co-presheaf $\Gamma \in \mathcal{P}_S$:

$$
\Delta_\Gamma : \mathcal{P}_S \to \mathcal{P}_S^\Gamma \\
\Delta_\Gamma A (U, \gamma) := AU \\
\Delta_\Gamma : \mathcal{N}_S \to \mathcal{N}_S^\Gamma \\
\Delta_\Gamma X (U, \gamma) := XU
$$

$$
\Delta_\bar{\Gamma} : \bar{\mathcal{P}}_S \to \bar{\mathcal{P}}_S^\Gamma \\
\Delta_\bar{\Gamma} X (U, \gamma) := XU \\
\Delta_\bar{\Gamma} : \bar{\mathcal{Q}}_S \to \bar{\mathcal{Q}}_S^\Gamma \\
\Delta_\bar{\Gamma} A (U, \Gamma, H) := A(U, H)
$$

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

$$
\text{hom}_{\mathcal{P}_S}(A, B)_\Gamma := \text{hom}_{\mathcal{P}_S}(\Delta_\Gamma A, \Delta_\Gamma B) \\
\text{hom}_{\mathcal{Q}_S}(X, Y)_\Gamma := \text{hom}_{\mathcal{Q}_S}(\Delta_\bar{\Gamma} X, \Delta_\bar{\Gamma} Y)
$$

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

Let $S$ be a small, set-reflective, guarded subuniverse of a guarded Martin-Löf universe $U$ containing $\text{Inj}$. We can compose the enriched adjunctions obtained in Lemma 3.18 to obtain a single enriched adjunction $F \dashv U : N_S \to \bar{P}_S$, setting $F : \equiv L \circ \exists \bar{\pi}_H \circ \Delta \pi_H$ and $U : \equiv \forall \pi_H \circ \Delta \bar{\pi}_H \circ R$ as depicted in Figure 4. We will write $\text{ret}$ 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.

\textbf{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:

\[
FAU = \bigotimes_{H : \bar{\mathcal{F}}_S} \bigoplus f : \text{hom}_{\mathcal{E}_S}(U, \pi_{\pi_H}) A(\pi_{\pi_H})
\]

\[
UXU = \Pi_{H : \bar{\mathcal{F}}_S} \Pi f : \text{hom}_{\mathcal{E}_S}(U, \pi_{\pi_H}) R(X(\pi_{\pi_H}))
\]

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

\[
IOAU = \Pi_{H : \bar{\mathcal{F}}_S} \Pi f : \text{hom}_{\mathcal{E}_S}(U, \pi_{\pi_H}) RLO \bigotimes_{H' : \bar{\mathcal{F}}_S} \bigoplus f : \text{hom}_{\mathcal{E}_S}(\pi_{\pi_H}, \pi_{\pi_{H'}}) A(\pi_{\pi_{H'}})
\]

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

3.2.3 The model of univalent reference types

We have now defined all the basic elements of our model.
Towards univalent reference types

3.2.3.1 General recursion and stepping

Let $S$ be a small, set-reflective, guarded subuniverse of a guarded Martin-Löf universe $\mathcal{U}$ containing $\text{Inj}$. Abstract steps in our model are encoded in terms of a global element $\text{step} : \text{IO} 1$, defined using the guarded domain structure of $\text{IO} 1$ as $\text{step} \equiv \vartheta_{\text{IO} 1}(\text{next}(\text{ret} \ast))$.

**Lemma 3.22.** For any $u : \text{IO} A$, we have $\text{step} ; u = \vartheta_{\text{IO} A}(\text{next} u)$.

**Proof.** By unfolding the definition of the $\triangleright$-algebra structure pointwise in the model. ▶

Using $\text{fix}_\triangleright$, we can define a monadic fixed point combinator satisfying the equation $\text{rec} a = \text{step} ; h (\text{rec} h) a$.

$$
\text{rec} : ((A \to \text{IO} B) \to A \to \text{IO} B) \to A \to \text{IO} B
$$

$$
\text{rec} h \equiv \text{fix}_\triangleright (\lambda f. \lambda x. \vartheta_{\text{IO} B}(\text{next}[g \leftarrow f]. h g x))
$$

**Lemma 3.23.** We have $\text{rec} h a = \text{step} ; h (\text{rec} h) a$.

**Proof.** We compute as follows:

$$
\text{rec} h a
$$

by unfolding definitions

$$
\equiv \text{fix}_\triangleright (\lambda f. \lambda x. \vartheta_{\text{IO} B}(\text{next}[g \leftarrow f]. h g x)) a
$$

by $\text{fix}_\triangleright$ computation rule

$$
\equiv \vartheta_{\text{IO} B}(\text{next}[g \leftarrow \text{next}(\text{rec} h)]. h g a)
$$

by rules of delayed substitutions

$$
\equiv \vartheta_{\text{IO} B}(\text{next}(h (\text{rec} h) a))
$$

by Lemma 3.22

$$
= \text{step} ; h (\text{rec} h) a
$$

We are done. ▶

3.2.3.2 Store operations: getting, setting, and allocation

Let $S$ be a small, $\Sigma$-closed, set-reflective, guarded subuniverse of a guarded Martin-Löf universe $\mathcal{U}$ containing $\text{Inj}$. In this section, we explicitly construct the store operations in $\mathcal{P}_S$, which we summarize in Figure 5.

**Construction 3.24.** (Detailed construction of the getter). For any $A \in \mathcal{P}_S$, we can interpret the getter as a natural transformation $\text{get} : \text{IORef} A \to \text{IO} A$ in $\mathcal{P}_S$. Because the definition is a little subtle, we will do it step-by-step.

$$
\text{get} : \text{IORef} A \to \text{IO} A
$$

$$
\text{get}_U(\ell : [U], \phi : \triangleright [X \leftarrow \partial_U \ell], [X]_U = [A]_U) H f \equiv ? : \text{FA}(\pi_{\text{ret}} H)
$$

We proceed by based path induction on the singleton $(\partial_U, \partial f : \partial_{\pi_{\text{ret}} H} \circ f = \partial_U)$, setting $U \equiv ([U], \partial_{\pi_{\text{ret}} H} \circ f)$ and $f \equiv ([f], \text{refl})$.

$$
\text{get}_U(\ell : [U], \phi : \triangleright [X \leftarrow \partial_U \ell], [X]_U = [A]_U) V (f \equiv ([f], \text{refl})) \equiv ? : \text{FA}(\pi_{\text{ret}} H)
$$

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

$$
\text{get}_U(\ell : [U], \phi : \triangleright [X \leftarrow \partial_U \ell], [X]_U = [A]_U) H (f \equiv ([f], \text{refl})) \equiv
$$

$$
\partial_{\text{FA}(\pi_{\text{ret}} H)} ? : \triangleright \text{FA}(\pi_{\text{ret}} H)
$$
get : !Ref A → IO A
get_U(ℓ : [U], φ : ▶[X ← ιU.ℓ], [X]_U = [A]_U) H (f ≡ ([f], refl)) :=
θ_{F A(π, H)} next[X ← ιU.ℓ, ψ ← φ, x ← H @ [f] ℓ].
η^H(1, π, H, ψ, x)

set : !Ref A × A → IO 1
set_U(ℓ, φ, a) H (f ≡ ([f], refl)) :=
let H_{a/ℓ} := H[[f] ℓ ← next[X ← ιU.ℓ, ψ ← φ].ψ^{-1}(A a)] in
now (η^H(H_{a/ℓ}, 1, π, H, *))

alloc : A → IO (!Ref A)
alloc_U a H (f ≡ ([f], refl)) :=
let H_a := \begin{align*}
\text{inr} * & \rightarrow \text{next}(A \text{ inr} a) \\
\text{inl} ℓ & \rightarrow \text{next}(A \text{ inl} x)
\end{align*}
in
now (η^H(H_a, \text{inl}, \text{inr} *, \text{next refl}))

---

**Figure 5** Summary of the store operations in \( P_S \) when \( S \) is a small, \( Σ \)-closed, set-reflective, guarded subuniverse of a guarded Martin-Löf universe \( U \) containing \( \text{Inj} \).

Using the introduction rule for the later modality, we may unwrap the delayed identification \( φ \) to assume \( ψ : [X]_U = [A]_U \), as well as the delayed element \( H @ [f] ℓ \) to assume \( x : X(π H) \):

\[
\text{get}_U(ℓ : [U], φ : ▶[X ← ιU.ℓ], [X]_U = [A]_U) H (f ≡ ([f], refl)) :=
θ_{F A(π, H)} next[X ← ιU.ℓ, ψ ← φ, x ← H @ [f] ℓ].
η^H(?, 1, π, H, {?})
\]

Applying the unit of the reflection \( π : U \rightarrow \text{Set}_S \) and splitting the resulting goal, we have three holes:

\[
\text{get}_U(ℓ : [U], φ : ▶[X ← ιU.ℓ], [X]_U = [A]_U) H (f ≡ ([f], refl)) :=
θ_{F A(π, H)} next[X ← ιU.ℓ, ψ ← φ, x ← H @ [f] ℓ].
η^H(?, ? : hom_{\text{Set}_S}(π H, π H), {?}_2 : A(π H))
\]

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.

\[
\text{get}_U(ℓ : [U], φ : ▶[X ← ιU.ℓ], [X]_U = [A]_U) H (f ≡ ([f], refl)) :=
θ_{F A(π, H)} next[X ← ιU.ℓ, ψ ← φ, x ← H @ [f] ℓ].
η^H(1, π, H, {? : A(π H)})
\]

Recall that we have an identification \( φ : [X]_U = [A]_U \) in the type \( \text{Set}_S U \) of co-presheaves on \( U/ξ_S \); transporting by this identification in the family \( Z : [\text{Set}_S U] U \vdash Z(π H) f : \text{Set}_S \), we have a mapping from \( X(π H) H \) to \( A(π H) H \). We use this to fill the final hole:

\[
\text{get}_U(ℓ : [U], φ : ▶[X ← ιU.ℓ], [X]_U = [A]_U) H (f ≡ ([f], refl)) :=
θ_{F A(π, H)} next[X ← ιU.ℓ, ψ ← φ, x ← H @ [f] ℓ].
η^H(H, 1, π, H, ψ, x)
\]
Towards univalent reference types

This completes the definition of the getter.

\[\textbf{Construction 3.25 (Detailed construction of the setter).} \] For each \(A \in \mathcal{P}_S\), we define the setter as a natural transformation \(\text{IORef} A \times A \to \text{IO} \mathbf{1}\) in \(\mathcal{P}_S\).

\[
\begin{align*}
\text{set} : & \text{IORef} A \times A \to \text{IO} \mathbf{1} \\
\text{set}_U((\ell, \phi), a) \, H (f \equiv (|f|, \text{refl}))) := & \ ?_1 (\pi_3 H)
\end{align*}
\]

We apply the unit now of the lifting monad, followed by the unit of the reflection \(\circ : \mathcal{U} \to \text{Set}_S\), and then split the goal:

\[
\begin{align*}
\text{set} : & \text{IORef} A \times A \to \text{IO} \mathbf{1} \\
\text{set}_U((\ell, \phi), a) \, H (f \equiv (|f|, \text{refl}))) := \\
\text{now} (\eta^{\circ}_U (\ ?_0 : \mathcal{P}_S, \ ?_1 : \text{home}_S(\pi_3 H, \pi_3 ?_0), *))
\end{align*}
\]

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 \(?_1 \equiv 1_{\pi_3 H}\), completing the definition of the setter as follows:

\[
\begin{align*}
\text{set} : & \text{IORef} A \times A \to \text{IO} \mathbf{1} \\
\text{set}_U((\ell, \phi), a) \, H (f \equiv (|f|, \text{refl}))) := \\
\text{let} \ H_{a/\ell} := H[|f| / \ell \mapsto \text{next}X \leftarrow \partial_U \ell, \psi \leftarrow \phi, \psi^{-1}(Af a)] \text{ in} \\
\text{now} (\eta^{\circ}(H_{a/\ell}, 1_{\pi_3 H}, *))
\end{align*}
\]

\[\textbf{Construction 3.26 (The allocator).} \] For each \(A \in \mathcal{P}_S\), we define the allocator as a natural transformation \(A \to \text{IO} (\text{IORef} A)\) in \(\mathcal{P}_S\).

\[
\begin{align*}
\text{alloc} : & A \to \text{IO} (\text{IORef} A) \\
\text{alloc}_U a \, H (f \equiv (|f|, \text{refl}))) := \\
\text{let} \ H_a := & \begin{cases} 
\text{inl} \ell \mapsto \text{next}X \leftarrow \partial_{\pi_3 H} \ell, x \leftarrow H \oplus \ell, X \text{inl} x \quad & \text{in} \\
\text{inr} * \mapsto \text{next}(A \text{inr} a) \quad & \text{now} (\eta^{\circ}(H_a, \text{inl}, (\text{inr} *, \text{next refl})))
\end{cases}
\end{align*}
\]

Above, we have defined a new heap \(H_a\) whose underlying finite set of locations is the coproduct \(|\pi_3 H| + 1\), filling the new location with \(a\) and return the pointer to this location.

3.2.3.3 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.

\[\textbf{Theorem 3.27.} \] Let \(\mathcal{S}\) be a small, \(\Sigma\)-closed, set-reflective, guarded subuniverse of a guarded Martin-Löf universe \(\mathcal{U}\) containing \(\text{Inj}\) 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 $\mathcal{P}_S = \text{Fun}(\mathcal{C}_S, \text{Set}_S)$;
2. the reference type connective is interpreted as in Construction 3.13;
3. the computational monad is given by $\text{IO} = U \circ F$ as defined in Figure 4;
4. general recursion and the store operations are interpreted as in Sections 3.2.3.1 and 3.2.3.2.

Proof. We note that $\mathcal{P}_S = \text{Fun}(\mathcal{C}_S, \text{Set}_S)$ is locally cartesian closed in spite of the fact that $\mathcal{C}_S$ is as large as $\text{Set}_S$ is: local cartesian closure nonetheless follows because $\text{Set}_S$ is reflective in $\mathcal{U}$ and $\mathcal{C}_S$ 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 3.27) is contingent on there existing a model of guarded homotopy type theory in which there can be found a suitably small, $\Sigma$-closed, set-reflective, guarded subuniverse of a guarded Martin-Löf universe containing $\text{Inj}$. 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 $\omega$. 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 $S \subseteq \mathcal{U}$ of a guarded Martin-Löf universe $\mathcal{U}$ containing $\text{Inj}$.

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.
Towards univalent reference types

1. Sterling et al. [43] have given non-univalent denotational semantics of polymorphic \( \lambda \)-calculus with recursive types and general reference types. It is within reach to adapt this model to the univalent setting, obtaining even more program equivalences 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


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.


Towards univalent reference types


