Partiality, Revisited

The Partiality Monad as a Quotient Inductive-Inductive Type

Thorsten Altenkirch[1(✉)], Nils Anders Danielsson[2(✉)], and Nicolai Kraus[1(✉)]

[1] University of Nottingham, Nottingham, UK
{thorsten.altenkirch,nicolai.kraus}@nottingham.ac.uk
[2] University of Gothenburg, Gothenburg, Sweden
nad@cse.gu.se

Abstract. Capretta's delay monad can be used to model partial computations, but it has the "wrong" notion of built-in equality, strong bisimilarity. An alternative is to quotient the delay monad by the "right" notion of equality, weak bisimilarity. However, recent work by Chapman et al. suggests that it is impossible to define a monad structure on the resulting construction in common forms of type theory without assuming (instances of) the axiom of countable choice.

Using an idea from homotopy type theory—a higher inductive-inductive type—we construct a partiality monad without relying on countable choice. We prove that, in the presence of countable choice, our partiality monad is equivalent to the delay monad quotiented by weak bisimilarity. Furthermore we outline several applications.

1 Introduction

Computational effects can be modelled using monads, and in some functional programming languages (notably Haskell) they are commonly used as a program structuring device. In the presence of dependent types one can both write and reason about monadic programs. From a type theorist's point of view, even a "pure" functional language like Haskell is not really pure as it has built-in effects, one of which is partiality: a function does not necessarily terminate. It is thus natural to look for a partiality monad which makes it possible to model partial computations and to reason about possibly non-terminating programs.

Capretta modeled partial computations using a coinductive construction that we call the *delay monad* [6]. We use the notation $\mathrm{D}(A)$ for Capretta's type of delayed computations over a type A. $\mathrm{D}(A)$ is coinductively generated by $\mathsf{now} : A \to \mathrm{D}(A)$ and $\mathsf{later} : \mathrm{D}(A) \to \mathrm{D}(A)$. Examples of elements of $\mathrm{D}(A)$ include $\mathsf{now}(a)$ and $\mathsf{later}(\mathsf{later}(\mathsf{now}(a)))$, as well as the infinitely delayed value $\bot$, defined

T. Altenkirch—Supported by EPSRC grant EP/M016994/1 and by USAF, Airforce office for scientific research, award FA9550-16-1-0029.

N.A. Danielsson—Supported by a grant from the Swedish Research Council (621-2013-4879).

N. Kraus—Supported by EPSRC grant EP/M016994/1.

J. Esparza and A.S. Murawski (Eds.): FOSSACS 2017, LNCS 10203, pp. 534–549, 2017.
DOI: 10.1007/978-3-662-54458-7_31

by the guarded equation $\bot = \mathsf{later}(\bot)$. We can model recursive programs as Kleisli arrows $A \to \mathrm{D}(B)$, and we can construct fixpoints of (ω-continuous) functions of type $(A \to \mathrm{D}(B)) \to (A \to \mathrm{D}(B))$, see Benton et al. [5].

Unfortunately, Capretta's delay monad is sometimes too intensional. It is often appropriate to treat two computations as equal if they terminate with the same value, but the delay monad allows us to count the number of "steps" (later constructors) used by a computation.

Capretta addressed this problem by defining a relation that we call *weak bisimilarity*, $\sim_{\mathrm{D}}$, and that relates expressions that only differ by a finite number of later constructors [6]. Capretta proved that the delay monad combined with weak bisimilarity is a monad in the category of setoids.

A setoid is a pair consisting of a type and an equivalence relation on that type. Setoids are sometimes used to approximate quotient types in type theories that lack support for quotients. However, a major difference between setoids and quotient types is that setoids do not provide a mechanism for information hiding. Using the setoid approach basically boils down to introducing a new relation together with the convention that all constructions have to respect this relation. A problem with this approach is that it can lead to something which has informally become known as *setoid hell*, in which one is forced to prove that a number of constructions—even some that do not depend on implementation details by, say, pattern matching on the now and later constructors—preserve setoid relations. This kind of problem does not afflict quotient types.

In a type theory with quotient types [14], one can consider using the quotient $\mathrm{D}(A)/{\sim_{\mathrm{D}}}$ as the type of partial computations of type A. This idea was discussed in a talk by Uustalu [7], reporting on joint work with Capretta and the first-named author of the current paper. However, the idea does not seem to work as intended. It is an open problem—and believed to be impossible—to show that this construction actually constitutes a monad (in "usual" forms of type theory).

With an additional assumption, Chapman et al. have managed to show that the partiality operator $\mathrm{D}(-)/{\sim_{\mathrm{D}}}$ is a monad [8]. This additional assumption is known as *countable choice*. To express what this is, first note that the *propositional truncation*, written $\|-\|$ and sometimes called "squashing", is an operation that turns a type into a proposition (a type with at most one element). We can see $\|A\|$ as the quotient of A by the total relation. Countable choice says that Π and $\|-\|$ commute if the domain is the natural numbers, in the sense that there is a function from $\Pi_{n:\mathbb{N}} \|P(n)\|$ to $\|\Pi_{n:\mathbb{N}} P(n)\|$. Even though this principle holds in some models, its status in type theory is unclear: the principle is believed to be independent of several variants of type theory. Recently Coquand et al. have shown that it cannot be derived in a theory with propositional truncation and a single univalent universe [10], speculating that the result might extend to a theory with a hierarchy of universes. Furthermore Richman argues that countable choice should be avoided in constructive reasoning [18]. The main purpose of the present paper is to define a partiality monad without making use of this principle.

The situation with the quotiented delay monad is similar to that of one variant of the real numbers in constructive mathematics. If the Cauchy reals are

defined as a quotient, then it is impossible to prove a specific form of the statement that every Cauchy sequence of Cauchy reals has a limit using $\mathsf{IZF}_{\mathrm{Ref}}$, a constructive set theory without countable choice [17]. It is suspected that corresponding statements are also impossible to prove in several variants of type theory. An alternative solution was put forward in the context of homotopy type theory [20]. In that approach, the reals are constructed inductively simultaneously with a notion of closeness, and the quotienting is done directly in the definition using a *higher inductive-inductive type* (HIIT).

In 2015, Andrej Bauer and the first-named author of the current paper suggested to use a similar approach to define a partiality monad without using countable choice, an idea which was mentioned by Chapman et al. [8]. Here, we show that this is indeed possible.

We do not make use of the full power of HIITs, but restrict ourselves to *set-truncated* HIITs. We call such types *quotient inductive-inductive types*, *QIITs*, following Altenkirch and Kaposi [1]. Some of the theory of QIITs is developed in the forthcoming PhD thesis of Dijkstra [12], see also Altenkirch et al. [2]. Although type theory extended with QIITs is still experimental and currently lacks a solid foundation, QIITs are a significantly simpler concept than full-blown HIITs. It is conjectured that QIITs exist in some computational models of type theory.

The type theory that we work in can be described as a fragment of the theory considered in the standard textbook on homotopy type theory [20] (henceforth referred to as the HoTT book), and is quite close to the theory considered by Chapman et al. [8]. Details are given in Sect. 2. The construction of our partiality monad is given in Sect. 3, together with its elimination principle and some properties. Furthermore we show that it gives us *free ω-cpos* in a sense that we will make precise. In Sect. 4 we show that, assuming countable choice, our partiality monad is equivalent to (in bijective correspondence to) the one of Chapman et al. [8]. We outline some applications of the partiality monad in Sect. 5, and conclude with a short discussion in Sect. 6.

Agda Formalisation. The paper is accompanied by a formal development [3] in Agda.

At the time of writing, Agda does not directly support QIITs. We have chosen to represent them by postulating their elimination principles together with the equalities they are supposed to satisfy. In some cases (but not for the partiality monad) we have also made use of Agda's experimental rewriting feature [9] to turn postulated equalities into judgmental computation rules.

Note that there are differences between the Agda code and the presentation in the text. For one, the formalisation discusses various additional topics that have been omitted in the paper for reasons of space, and is more rigorous. Furthermore, the paper defines the partiality monad's elimination principle as a universal property. In the formalisation the elimination principle is given as an induction principle, but we also prove that this principle is interderivable with the universal property. Finally there are a number of small differences between

the formalisation and the text, and some results in the paper have not been formalised at all, most notably the results about the reals in Sect. 5.2.

2 Background: Type Theory with Quotient Inductive-Inductive Types

We work in intensional type theory of Martin-Löf style with all the usual components (e.g. Π, Σ, inductive types), including the identity type (we use the notation $x = y$). We assume that equality of functions is extensional, and that (strong) bisimilarity implies equality for coinductive types.

Chapman et al. [8] assume the axiom of *uniqueness of identity proofs*, UIP, for all small types (types in the lowest universe). UIP holds for a type A if, for any elements x, $y : A$, if we have equalities p, $q : x = y$, then we have $p = q$. Instead of postulating an axiom, we prefer to work in a more general setting and restrict ourselves to types with the corresponding property. This approach is compatible with homotopy type theory. In the language of homotopy type theory, we work with *sets* or 0-*truncated types*; a type is a set if and only if it satisfies UIP. When we write $A : \mathsf{Set}$, we mean that A is a type (in some universe) with the property of being a set; and when we write $B : A \to \mathsf{Set}$, we mean that B is a family of types such that each $B(a)$ is a set.

Similarly to $A : \mathsf{Set}$, we write $P : \mathsf{Prop}$ for a type P with the property that it is a proposition, i.e. a (-1)-truncated type, i.e. a type with the property that any two of its elements are equal. A proposition is also a set. The type of all propositions in a certain universe is closed under all operations that are relevant to us, and the same applies to sets.

In addition to UIP, Chapman et al. [8] assume *propositional extensionality*—that logically equivalent propositions are equal—for all small propositions. This property is equivalent to the *univalence axiom* [20], restricted to (small) propositions. Just like Chapman et al., we only require propositional extensionality (not full univalence) for our development, with the exception that univalence is used to show that certain precategories (in the sense of the HoTT book [20]) are categories. For an example of how propositional extensionality is used, see Lemma 7.

Chapman et al. [8] also assume the existence of quotient types in the style of Hofmann [14]. Given a set A and a propositional relation $\sim$ on it, the (set-) quotient $A/\!\sim$ can in homotopy type theory be constructed as a higher inductive type with three constructors [20]:

$$\begin{aligned} [-] &: A \to A/\!\sim \\ [-]^{=} &: \Pi_{a,b:A}\, a \sim b \to [a] = [b] \\ \mathsf{irr} &: \Pi_{x,y:A/\sim}\, \Pi_{p,q:x=y}\, p = q \end{aligned}$$

The last constructor irr ensures that any two parallel equalities are equal, that is, that $A/\!\sim$ is set-truncated. We call a higher inductive type with such a set-truncation constructor a *quotient inductive type* (QIT).

As noted above, Chapman et al. [8] use countable choice, which we want to avoid. Instead we make use of quotient *inductive-inductive* types (QIITs) [2,12]. From the point of view of homotopy type theory, these are set-truncated higher inductive-inductive types (HIITs); some other examples of HIITs can be found in the HoTT book [20, Chap. 11]. While it seems plausible that QIITs exist in some computational models of type theory, this has yet to be determined.

3 The Partiality Monad

As indicated in the introduction we define the partiality monad $(-)_\perp$ as a QIIT, defining the type $A_\perp$ simultaneously with an ordering relation $\sqsubseteq$ on $A_\perp$. We will first describe the constructors and the elimination principle of this definition, and then show that $A_\perp$ is the underlying type of the free ω-cpo (see Definition 4) on A, thus proving that $(-)_\perp$ is a monad. In the final part of this section we will give a characterisation of the ordering relation; this is perhaps not as trivial as one might expect, given the relation's definition.

Note that our construction of $A_\perp$ can be seen as a further example of a free algebraic structure defined in type theory. It was discussed in the HoTT book [20, Chap. 6.11] that free groups can be defined as (in our terminology) quotient inductive types, while it is well-known that even simpler examples can be defined as ordinary inductive types.

3.1 The Definition and Its Elimination Principles

Let A be a set. We define the set $A_\perp$ simultaneously with a binary propositional relation on $A_\perp$, written $\sqsubseteq$. The set $A_\perp$ is generated by the following four constructors, plus a set-truncation constructor:

$$\begin{aligned} \eta &: A \to A_\perp & \bigsqcup &: (\Sigma_{s:\mathbb{N}\to A_\perp} \Pi_{n:\mathbb{N}}\, s_n \sqsubseteq s_{n+1}) \to A_\perp \\ \perp &: A_\perp & \alpha &: \Pi_{x,y:A_\perp}\, x \sqsubseteq y \to y \sqsubseteq x \to x = y \end{aligned}$$

The constructor η tells us that any element of A can be viewed as an element of $A_\perp$, and $\perp$ represents a non-terminating computation. The constructor $\bigsqcup$ is intended to form least upper bounds of increasing sequences, and α ensures that the ordering relation $\sqsubseteq$ is antisymmetric.

The relation $\sqsubseteq$ is a type family that is indexed twice by $A_\perp$. It is generated by six constructors. One of these constructors says that, for any $x, y : A_\perp$, the type $x \sqsubseteq y$ is a proposition ($\Pi_{p,q:x\sqsubseteq y}\, p = q$). Because any two proofs of $x \sqsubseteq y$ are equal, we do not name the constructors of the ordering relation. The remaining constructors are given as inference rules, where each rule is implicitly Π-quantified over its unbound variables (the same comment applies to other definitions below):

$$\frac{}{x \sqsubseteq x} \qquad \frac{x \sqsubseteq y \qquad y \sqsubseteq z}{x \sqsubseteq z} \qquad \frac{}{\perp \sqsubseteq x} \qquad \frac{}{\Pi_{n:\mathbb{N}}\, s_n \sqsubseteq \bigsqcup(s,p)} \qquad \frac{\Pi_{n:\mathbb{N}}\, s_n \sqsubseteq x}{\bigsqcup(s,p) \sqsubseteq x}$$

The rules state that $\sqsubseteq$ is reflexive and transitive, that $\bot$ is at least as small as any other element of $A_\bot$, and that $\bigsqcup$ constructs least upper bounds.

Now we will give the elimination principle of $(-)_\bot$ and $\sqsubseteq$. This principle can be stated in different ways. One way would be to state it as an induction principle, along the following lines: *Given a family* $P : A_\bot \to \mathsf{Set}$ *and [... something for* $\sqsubseteq$*...], and given elements of* $P(\bot)$, $\Pi_{a:A} P(\eta(a))$, *[... and so on...], we can conclude that* $\Pi_{x:A_\bot} P(x)$ *and [...].* We take this approach in our formalisation; for another example, see the presentation of the Cauchy reals in the HoTT book [20, Chap. 11.3.2]. However, because the two types are defined simultaneously and involve constructors targeting the equality type, the induction principle may look somewhat involved and perhaps even ad-hoc, and it may not be obvious that it is the "correct" one.

Instead, we present a universal property. Dijkstra [12] and Altenkirch et al. [2] have worked out a general form and rules for a large class of quotient inductive-inductive types. In their setting, any specification of a QIIT gives rise to a *category of algebras*, following methods that have been used for W-types [4] and certain higher inductive types [19], and if this category has a (homotopy-) initial object, then this object is taken as the definition of the QIIT. We use the following algebras:

Definition 1 (partiality algebras). *A* partiality algebra *over the set* A *consists of a set* X*; a propositional binary relation on* X, $\sqsubseteq_X$*; an element* $\bot_X : X$*, a family* $\eta_X : A \to X$*, and a family* $\bigsqcup_X : (\Sigma_{s:\mathbb{N}\to X} \Pi_{n:\mathbb{N}}\, s_n \sqsubseteq_X s_{n+1}) \to X$*; and the following laws:*

$$x \sqsubseteq_X x \qquad x \sqsubseteq_X y \to y \sqsubseteq_X z \to x \sqsubseteq_X z$$

$$\bot_X \sqsubseteq_X x \qquad x \sqsubseteq_X y \to y \sqsubseteq_X x \to x = y$$

$$\Pi_{n:\mathbb{N}}\, s_n \sqsubseteq_X \bigsqcup_X (s,p) \qquad (\Pi_{n:\mathbb{N}}\, s_n \sqsubseteq_X x) \to \bigsqcup_X (s,p) \sqsubseteq_X x$$

The type X *and the type family* $\sqsubseteq_X$ *are allowed to target universes distinct from the one that* A *lives in.*

For two partiality algebras over the same set A, $(X, \sqsubseteq_X, \bot_X, \eta_X, \bigsqcup_X)$ *and* $(Z, \sqsubseteq_Z, \bot_Z, \eta_Z, \bigsqcup_Z)$*, a* morphism of partiality algebras *from the former to the latter consists of a function* $f : X \to Z$ *satisfying the following laws: First,* f *has to respect the ordering relation,* $f^\sqsubseteq : x \sqsubseteq_X y \to f(x) \sqsubseteq_Z f(y)$*. Second,* f *has to preserve some of the constructors,* $f(\bot_X) = \bot_Z$, $f \circ \eta_X = \eta_Z$, *and* $f(\bigsqcup_X (s,p)) = \bigsqcup_Z (f \circ s, f^\sqsubseteq \circ p)$.

Let us denote this structure of objects and morphisms by Part_A.

The structure Part_A is a category, in which the identity morphism is the identity function, and composition of morphisms is composition of functions.

We can now make the elimination principle precise. Note that the tuple $(A_\bot, \sqsubseteq, \bot, \eta, \bigsqcup)$ is a partiality algebra. As the *elimination principle of* $A_\bot$ *and* $\sqsubseteq$ we take the statement that there is a unique (up to equality) morphism from this partiality algebra to any other partiality algebra over A. In the terminology of the HoTT book [20], the statement that there is a morphism is basically the

recursion principle of $(-)_\perp$ and $\sqsubseteq$, while uniqueness gives us the power of the induction principle (with *propositional* computation rules). Note that allowing the type X and the type family $\sqsubseteq_X$ to target arbitrary universes enables us to make use of large elimination.

We do not lose anything by using a universal property instead of an induction principle, at least for the induction principle referred to in the following theorem. The theorem is similar to results due to Dijkstra [12] and Altenkirch et al. [2]. It is stated without proof here, but a full proof of the fact can be found in our Agda development.

Theorem 2. *The elimination principle of $A_\perp$ and $\sqsubseteq$ can be stated as an induction principle. This induction principle, which comes with propositional rather than definitional computation rules, is interderivable with the universal property given above.*

Note that we could have defined $A_\perp$ and $\sqsubseteq$ differently. For instance, we could have omitted the set-truncation constructor from the definition of $A_\perp$, and then proved that the type is a set, following the approach taken for the Cauchy reals in the HoTT book [20]. However, if we had done this, then our definitions would have been less close to the general framework mentioned above [2,12].

As a simple demonstration of the universal property we construct an induction principle for $A_\perp$ that can be used when eliminating into a proposition. Following the terminology of the HoTT book [20, Chap. 11.3.2], we call it *partiality induction*:

Lemma 3. *Let P be a family of propositions on $A_\perp$ such that both $P(\perp)$ and $\Pi_{a:A}\, P(\eta(a))$ hold. Assume further that, for any increasing sequence $s : \mathbb{N} \to A_\perp$ (with corresponding proof p), $\Pi_{n:\mathbb{N}}\, P(s_n)$ implies $P(\bigsqcup(s, p))$. Then we can conclude $\Pi_{x:A_\perp}\, P(x)$.*

Proof. The proof uses a standard method. We define a partiality algebra where the set is $Z :\equiv \Sigma_{x:A_\perp}\, P(x)$; the binary relation is $\sqsubseteq$, ignoring the second projections of the values in Z; and the rest of the algebra is constructed using the assumptions. The universal property gives us a morphism m from the initial partiality algebra to this one, and in particular a function of type $A_\perp \to Z$. We are done if we can show that this function, composed with the first projection, is the identity on $A_\perp$. Note that the first projection can be turned into a partiality algebra morphism fst. Thus, by uniqueness, the composition of fst and m has to be the unique morphism from the initial partiality algebra to itself, and the function component of this morphism is the identity. □

3.2 ω-Complete Partial Orders

Another way of characterising our quotient inductive-inductive partiality monad is to say that $A_\perp$ is the *free (pointed) ω-cpo* over A:

Definition 4. *Let us denote the category $\mathsf{Part}_{\mathbf{0}}$, where $\mathbf{0}$ is the empty type, by ω-CPO. An ω-cpo is an object of this category.*

Let us quickly check that this definition makes sense. A partiality algebra on $\mathbf{0}$ is a set X with a binary propositional relation $\sqsubseteq_X$ that is a partial order. There is a least element $\bot_X$ and any increasing sequence has a least upper bound. There is also a function of type $\mathbf{0} \to X$, which we omit below as it carries no information.

We can now relate the category of sets [20, Example 9.1.7], written SET, to the category ω-CPO. For any ω-cpo we can take the underlying set, and it is easy to see that this yields a functor, in the sense of the HoTT book [20, Definition 9.2.1], $\mathsf{U} : \omega\text{-}\mathsf{CPO} \to \mathsf{SET}$.

We also have a functor $\mathsf{F} : \mathsf{SET} \to \omega\text{-}\mathsf{CPO}$, constructed as follows: The functor maps a set A to the ω-cpo $(A_\bot, \sqsubseteq, \bot, \bigsqcup)$. For the morphism part, assume that we have a function $f : A \to B$. Then $(B_\bot, \sqsubseteq, \bot, \eta \circ f, \bigsqcup)$ is an A-partiality algebra, and hence there is a morphism to this algebra from the initial A-partiality algebra $(A_\bot, \sqsubseteq, \bot, \eta, \bigsqcup)$. By removing the components $\eta \circ f : A \to B_\bot$ and $\eta : A \to A_\bot$ we get a morphism between ω-cpos.

The function η lifts to a natural transformation from the identity functor to $\mathsf{U} \circ \mathsf{F}$. In order to construct a natural transformation from $\mathsf{F} \circ \mathsf{U}$ to the identity functor, assume that we are given some ω-cpo X. We can construct an ω-cpo morphism from $\mathsf{F}(\mathsf{U}(X))$ to X by noticing that $(\mathsf{U}(X), \sqsubseteq_X, \bot_X, id, \bigsqcup_X)$ is a partiality algebra on $\mathsf{U}(X)$, and thanks to initiality we get a morphism m from $\mathsf{F}(\mathsf{U}(X))$ to X satisfying $m \circ \eta = id$. After proving some equalities we end up with the following result, where the definition of "adjoint" is taken from the HoTT book [20, Definition 9.3.1]:

Theorem 5. *For a given set A, the functor F is a left adjoint to the forgetful functor U. This means that $\mathsf{F}(A)$ can be seen as the* free *ω-cpo over A.* □

Thus we get a justification for calling the concept that we are discussing the partiality *monad*:

Corollary 6. *The composition $\mathsf{U} \circ \mathsf{F} : \mathsf{SET} \to \mathsf{SET}$, which maps objects A to $A_\bot$, is a monad.* □

Note that one can also construct a monad structure on $(-)_\bot$ directly. Let us fix the set A. The unit is given by η. For the multiplication $\mu : (A_\bot)_\bot \to A_\bot$, note that $A_\bot$ can be given the structure of a partiality algebra over $A_\bot$ in a trivial way: the underlying set is $A_\bot$, the function $\eta_{A_\bot} : A_\bot \to A_\bot$ is the identity, $\sqsubseteq_{A_\bot}$ is $\sqsubseteq$, and so on. This gives us the function μ as the unique morphism from the initial partiality algebra to this one. Proving the monad laws is straightforward.

3.3 A Characterisation of the Relation $\sqsubseteq$

To further analyse the QIIT construction, we show how the relation $\sqsubseteq$ on the set $A_\bot$ behaves.[1] These results are useful when working with the partiality monad, and will play an important role in the next section of the paper. The arguments

[1] The work presented in Sect. 3.3 was done in collaboration with Paolo Capriotti.

are only sketched here, details are given in the formalisation. We use the propositional truncation $\|-\|$ (also known as "squashing"), which turns a type into a proposition. It can be implemented by quotienting with the trivial relation.

We know that $\bot \sqsubseteq y$ is (by definition) satisfied for any $y : A_\bot$, and for the least upper bound we have that $\bigsqcup(s, q) \sqsubseteq y$ is equivalent to $\Pi_{n:\mathbb{N}}\, s_n \sqsubseteq y$. The following lemma provides a characterisation of $\eta(a) \sqsubseteq y$, for any $a : A$:

Lemma 7. *The binary relation $\sqsubseteq$ on $A_\bot$ has the following properties:*

$$\begin{aligned} \eta(a) &\sqsubseteq \bot && \leftrightarrow && \mathsf{0} \\ \eta(a) &\sqsubseteq \eta(b) && \leftrightarrow && a = b \\ \eta(a) &\sqsubseteq \bigsqcup(s, q) && \leftrightarrow && \|\Sigma_{n:\mathbb{N}}\, \eta(a) \sqsubseteq s_n\| \end{aligned}$$

We will give the proof of this lemma later and make a remark first. Constructors in "HIT-like" definitions, e.g. QIITs, may in general be neither injective nor disjoint. For instance, $\bigsqcup(\lambda n.\bot, q) = \bot$. However, we have the following lemma:

Corollary 8. *For any $a : A$ and $y : A_\bot$, we have that $\eta(a) \sqsubseteq y$ implies that $\eta(a) = y$. In particular, η is injective: if $\eta(a) = \eta(b)$, then $a = b$. Moreover, we have $\eta(a) \neq \bot$.*

Proof (of Corollary 8). The last two claims are simple consequences of the lemma and reflexivity. For the first claim, let us fix $a : A$ and apply Lemma 3 with $P(y) :\equiv \eta(a) \sqsubseteq y \to \eta(a) = y$. The only non-immediate step is the case for $\bigsqcup(s, q)$, where we can assume $\Pi_{n:\mathbb{N}}\, P(s_n)$. From $\eta(a) \sqsubseteq \bigsqcup(s, q)$ and Lemma 7 we get $\|\Sigma_{n:\mathbb{N}}\, \eta(a) \sqsubseteq s_n\|$. We are proving a proposition, so we can assume that we have $n : \mathbb{N}$ such that $\eta(a) \sqsubseteq s_n$. This implies that, for all $m \geq n$, $\eta(a) \sqsubseteq s_m$ and hence, by the "inductive hypothesis", $\eta(a) = s_m$. Thus $\eta(a)$ is an upper bound of s, so we get $\bigsqcup(s, q) \sqsubseteq \eta(a)$, which by antisymmetry implies $\bigsqcup(s, q) = \eta(a)$. □

The proof of the lemma is more technical. The approach is similar to that used to prove some results about the real numbers defined as a HIIT in the HoTT book [20, Theorems 11.3.16 and 11.3.32]. We only give a sketch here, the complete proof can be found in our Agda formalisation.

Proof (of Lemma 7). For every $a : A$ we construct a relation in $A_\bot \to \mathsf{Prop}$ by applying the elimination principle of $A_\bot$ and $\sqsubseteq$, treating Prop as a partiality algebra over A in the following way:

$$\begin{aligned} P \sqsubseteq_{\mathsf{Prop}} Q &:\equiv (P \to Q) & \eta_{\mathsf{Prop}}(b) &:\equiv (a = b) \\ \bot_{\mathsf{Prop}} &:\equiv \mathbf{0} & \bigsqcup_{\mathsf{Prop}}(S, P) &:\equiv \|\Sigma_{n:\mathbb{N}}\, S_n\| \end{aligned}$$

Propositional extensionality is used to prove that Prop is a set (this is a variant of an instance of Theorem 7.1.11 in the HoTT book [20]), and to prove the antisymmetry law.

Using Lemma 3 one can then show that the defined relation is pointwise equal to $\eta(a) \sqsubseteq -$, and it is easy to see that the relation has the properties claimed in the statement of Lemma 7. □

Using the results above one can prove that the order is flat, in the sense that if x and y are distinct from $\bot$ and $x \neq y$, then $x \not\sqsubseteq y$ (see the formalisation).

4 Relation to the Coinductive Construction

In this section we compare our QIIT to Capretta's coinductive delay monad [6], quotiented by weak bisimilarity [8]. Let us start by giving Capretta's construction, as already outlined in the introduction. $\mathcal{U}$ stands for a universe of types.

Definition 9 (delay monad and weak bisimilarity). *For a set A the* delay monad $\mathrm{D}(A)$ *is the coinductive type generated by* $\mathsf{now} : A \to \mathrm{D}(A)$ *and* $\mathsf{later} : \mathrm{D}(A) \to \mathrm{D}(A)$. *The "terminates with" relation* $\downarrow_{\mathrm{D}} : \mathrm{D}(A) \to A \to \mathcal{U}$ *is the indexed inductive type generated by two constructors of type* $\eta(a) \downarrow_{\mathrm{D}} a$ *and* $p \downarrow_{\mathrm{D}} a \to \mathsf{later}(p) \downarrow_{\mathrm{D}} a$. *Furthermore, x and $y : \mathrm{D}(A)$ are said to be* weakly bisimilar*, written $x \sim_{\mathrm{D}} y$, if $\Pi_{a:A}\, x \downarrow_{\mathrm{D}} a \leftrightarrow y \downarrow_{\mathrm{D}} a$ holds.*

It is easy to give $\mathrm{D}(A)$ the structure of a monad. Note that $x \downarrow_{\mathrm{D}} a$ can alternatively be defined to be $\Sigma_{n:\mathbb{N}}\, x = \mathsf{later}^n(\mathsf{now}(a))$. The types $x \downarrow_{\mathrm{D}} a$ and $x \sim_{\mathrm{D}} y$ are propositional, and $\sim_{\mathrm{D}}$ is an equivalence relation on $\mathrm{D}(A)$.

The goal of this section is to show that, in the presence of countable choice, the partiality monad $A_\bot$ is equivalent to $\mathrm{D}(A)/{\sim_{\mathrm{D}}}$. (We use the notion of equivalence from the HoTT book [20], which for sets is equivalent to bijective correspondence.) To understand the structure of the proof, let us observe that $\mathrm{D}(A)/{\sim_{\mathrm{D}}}$ is constructed as a "coinductive type that is quotiented afterwards", while $A_\bot$ is an "inductive type that is quotiented at the time of construction". To build a connection between these, it seems rather intuitive to consider an intermediate construction, either a "coinductive type that is quotiented at the time of construction" or an "inductive type that is quotiented afterwards". The theory of "higher coinductive types" has, as far as we know, not been explored much yet, so we go with the second option. We do not even need an *inductive* construction: it is well-known that coinductive structures can be represented using finite approximations, and here, it is enough to consider monotone functions. Thus, first we will show that $\mathrm{D}(A)$ is equivalent to a type of monotone sequences, carefully formulated, and that the equivalence lifts to the quotients. Then we will prove that, assuming countable choice, the quotiented monotone sequences are equivalent to $A_\bot$.

4.1 The Delay Monad and Monotone Sequences

For a set A we say that a function $g : \mathbb{N} \to A + \mathbf{1}$ is a monotone sequence if it satisfies the propositional property

$$\mathsf{ismon}(g) \;:\equiv\; \Pi_{n:\mathbb{N}}\, (g_n = g_{n+1}) + ((g_n = \mathsf{inr}(\star)) \times (g_{n+1} \neq \mathsf{inr}(\star)))\,.$$

The set of monotone sequences, $\Sigma_{g:\mathbb{N}\to A+\mathbf{1}}\, \mathsf{ismon}(g)$, is denoted by Seq_A. Below the notation $-_n$ will be used not only for functions, but also for monotone sequences; $(g, p)_n$ means g_n.

As Chapman et al. [8] observe, one can construct a sequence of type $\mathbb{N} \to A+\mathbf{1}$ from an element of $\mathrm{D}(A)$. If their construction is tweaked a little, then the resulting sequences are monotone, and the map is an equivalence:

Lemma 10. *The types* Seq_A *and* $\mathrm{D}(A)$ *are equivalent.*

Proof. We can simply give functions back and forth. Note that endofunctions on $\mathrm{D}(A)$ that correspond to later and "remove later, if there is one" can be mimicked for Seq_A: let us use the names *shift* and *unshift* : $\mathsf{Seq}_A \to \mathsf{Seq}_A$ for the functions that are determined by $\mathit{shift}(g)_0 :\equiv \mathsf{inr}(\star)$, $\mathit{shift}(g)_{n+1} :\equiv g_n$, and $\mathit{unshift}(g)_n :\equiv g_{n+1}$.

Define $j : \mathrm{D}(A) \to \mathsf{Seq}_A$ such that $j(\mathsf{now}(a))$ equals $\lambda n.\mathsf{inl}(a)$, and $j(\mathsf{later}(x))$ equals $\mathit{shift}(j(x))$, One way to do this is to define $j(z)_n$ by recursion on n, followed by case distinction on z. Furthermore, define $h : \mathsf{Seq}_A \to \mathrm{D}(A)$ in the following way: Given $s : \mathsf{Seq}_A$, do case distinction on s_0. If s_0 is $\mathsf{inl}(a)$, return $\mathsf{now}(a)$. Otherwise, return $\mathsf{later}(h(\mathit{unshift}(s)))$. It is straightforward to show that j and h are inverses of each other. □

As an aside, our formalisation shows that Lemma 10 holds even if A is not a set.

Next, we mimic the relation $\downarrow_{\mathrm{D}}$ by setting

$$\begin{aligned} &\downarrow_{\mathsf{Seq}} : \mathsf{Seq}_A \to A \to \mathcal{U} \\ &s \downarrow_{\mathsf{Seq}} a \;:\equiv\; \Sigma_{n:\mathbb{N}}\, s_n = \mathsf{inl}(a). \end{aligned}$$

The relation $\downarrow_{\mathsf{Seq}}$ is not in general propositional. To remedy this, we can truncate and consider $\|s \downarrow_{\mathsf{Seq}} a\|$. Using strategies explained by Kraus et al. [15], we have $\|s \downarrow_{\mathsf{Seq}} a\| \to s \downarrow_{\mathsf{Seq}} a$, so we can always extract a concrete value of n: a variant of the definition above in which the number n is required to be minimal is propositional, and this definition can be shown to be logically equivalent to both $\|s \downarrow_{\mathsf{Seq}} a\|$ and $s \downarrow_{\mathsf{Seq}} a$. See the formalisation for details.

We define the propositional relations $\sqsubseteq_{\mathsf{Seq}}$ and $\sim_{\mathsf{Seq}}$ by

$$\begin{aligned} s \sqsubseteq_{\mathsf{Seq}} t \;&:\equiv\; \Pi_{a:A}\, \|s \downarrow_{\mathsf{Seq}} a\| \to \|t \downarrow_{\mathsf{Seq}} a\| \text{ and} \\ s \sim_{\mathsf{Seq}} t \;&:\equiv\; s \sqsubseteq_{\mathsf{Seq}} t \times t \sqsubseteq_{\mathsf{Seq}} s. \end{aligned}$$

By checking that the equivalence from Lemma 10 maps $\sim_{\mathsf{Seq}}$-related elements to $\sim_{\mathrm{D}}$-related elements, we get:

Lemma 11. *The sets* $\mathsf{Seq}_A/{\sim_{\mathsf{Seq}}}$ *and* $\mathrm{D}(A)/{\sim_{\mathrm{D}}}$ *are equivalent.* □

4.2 Monotone Sequences and the QIIT Construction

As the final step of showing that $\mathrm{D}(A)/{\sim_{\mathrm{D}}}$ and $A_\perp$ are equivalent, we show that $\mathsf{Seq}_A/{\sim_{\mathsf{Seq}}}$ and $A_\perp$ are. The plan is as follows: There is a canonical function $w : \mathsf{Seq}_A \to A_\perp$ which can be extended to a function $\tilde{w} : \mathsf{Seq}_A/{\sim_{\mathsf{Seq}}} \to A_\perp$. The function $\tilde{w}$ is injective. Furthermore, if we assume countable choice, then the function w, and thus also $\tilde{w}$, are surjective. Thus $\tilde{w}$ is an equivalence.

Let us start by constructing w and $\tilde{w}$. We use a copairing function $[\eta \mid \bot] : A + \mathbf{1} \to A_\bot$ defined by $[\eta \mid \bot]\,(\mathsf{inl}(a)) :\equiv \eta(a)$ and $[\eta \mid \bot]\,(\mathsf{inr}(\star)) :\equiv \bot$, and define $w : \mathsf{Seq}_A \to A_\bot$ by $w(s,q) :\equiv \bigsqcup([\eta \mid \bot] \circ s, \ldots)$, with a canonical proof of monotonicity.

Lemma 12. *The function w is monotone: $\Pi_{s,t:\mathsf{Seq}_A}\, s \sqsubseteq_{\mathsf{Seq}} t \to w(s) \sqsubseteq w(t)$. Thus w extends to a map $\tilde{w} : \mathsf{Seq}_A/{\sim_{\mathsf{Seq}}} \to A_\bot$.*

Proof. For the second claim we show that w maps elements related by $\sim_{\mathsf{Seq}}$ to equal elements. This follows from the first claim by antisymmetry. For the first claim it suffices to find a function $k : \mathbb{N} \to \mathbb{N}$ such that, for all n, we have $[\eta \mid \bot]\,(s_n) \sqsubseteq [\eta \mid \bot]\,(t_{k(n)})$. Fix n. If s_n is $\mathsf{inr}(\star)$, then $[\eta \mid \bot]\,(s_n)$ is $\bot$, and $k(n)$ can thus be chosen arbitrarily. If s_n is $\mathsf{inl}(a)$, then we have $s \downarrow_{\mathsf{Seq}} a$ and therefore $t \downarrow_{\mathsf{Seq}} a$, which gives us a number $k(n)$ such that $t_{k(n)} = \mathsf{inl}(a)$ and $[\eta \mid \bot]\,(s_n) = \eta(a) = [\eta \mid \bot]\,(t_{k(n)})$. □

Lemma 13. *The function $\tilde{w}$ is injective: $\Pi_{s,t:\mathsf{Seq}_A/{\sim_{\mathsf{Seq}}}}\, \tilde{w}(s) = \tilde{w}(t) \to s = t$.*

Proof. It suffices to show that, for $s, t : \mathsf{Seq}_A$, $w(s) = w(t)$ implies $s \sim_{\mathsf{Seq}} t$. By symmetry, it is enough to fix $a : A$ and show $\|s \downarrow_{\mathsf{Seq}} a\| \to \|t \downarrow_{\mathsf{Seq}} a\|$, which follows from $s \downarrow_{\mathsf{Seq}} a \to \|t \downarrow_{\mathsf{Seq}} a\|$. If $s \downarrow_{\mathsf{Seq}} a$ then $w(s) = \eta(a)$, and thus also $w(t) = \eta(a)$. Using Lemma 7 and Corollary 8 we then get $\|\Sigma_{n:\mathbb{N}}\, \eta(a) = [\eta \mid \bot]\,(t_n)\|$, which implies $\|\Sigma_{n:\mathbb{N}}\, t_n = \mathsf{inl}(a)\|$. □

Lemma 14. *Under countable choice, w is surjective: $\Pi_{x:A_\bot}\, \|\Sigma_{s:\mathsf{Seq}_A}\, w(s) = x\|$.*

Proof. We apply the simplified induction principle presented in Lemma 3. The propositional predicate is $P(x) :\equiv \|\Sigma_{s:\mathsf{Seq}_A}\, w(s) = x\|$. Both $P(\bot)$ and $P(\eta(a))$ are trivial: in the first case we use the sequence that is constantly $\mathsf{inr}(\star)$, while in the second case we take the one that is constantly $\mathsf{inl}(a)$.

The interesting part is to show $P\,(\bigsqcup(f,p))$ for a given $f : \mathbb{N} \to A_\bot$ and $p : \Pi_{n:\mathbb{N}}\, f_n \sqsubseteq f_{n+1}$. By the mentioned induction principle, we can assume $\Pi_{n:\mathbb{N}}\, P(f_n)$, which unfolds to $\Pi_{n:\mathbb{N}}\, \|\Sigma_{t:\mathsf{Seq}_A}\, w(t) = f_n\|$. By countable choice, we can swap $\Pi_{n:\mathbb{N}}$ and $\|-\|$, which allows us to remove the truncation completely, because the goal is propositional. Hence we can assume $\Pi_{n:\mathbb{N}}\, \Sigma_{t:\mathsf{Seq}_A}\, w(t) = f_n$.

Using the usual distributivity law for Π and Σ (sometimes called the "type-theoretic axiom of choice"), we can assume that we are given $g : \mathbb{N} \to \mathsf{Seq}_A$ and a proof $\gamma : \Pi_{n:\mathbb{N}}\, w(g_n) = f_n$. By dropping the monotonicity proof and uncurrying, g gives us a function $g' : \mathbb{N} \times \mathbb{N} \to A + \mathbf{1}$ with the property that it assumes at most one value in A: If $g'_{i,j} = \mathsf{inl}(a)$, then (using γ) $\eta(a) \sqsubseteq f_i$, thus $\eta(a) \sqsubseteq \bigsqcup(f,p)$, and hence $\bigsqcup(f,p) = \eta(a)$ by Corollary 8. If we also have $g'_{k,m} = \mathsf{inl}(b)$, then $\eta(a) = \eta(b)$, which by Corollary 8 implies that $a = b$.

We use g' to construct an element of Seq_A. Take an arbitrary isomorphism $\sigma : \mathbb{N} \to \mathbb{N} \times \mathbb{N}$ (a split surjection would also be sufficient), and define a function $\tilde{g} : \mathbb{N} \to A + \mathbf{1}$ by

$$\tilde{g}(n) :\equiv \begin{cases} g'(\sigma_n), & \text{if } n = 0 \text{ or } g'(\sigma_n) \neq \mathsf{inr}(\star), \\ \tilde{g}(n-1), & \text{otherwise.} \end{cases}$$

The intuition is that $\tilde{g}(n)$ checks the first $n+1$ results of $g' \circ \sigma$ and chooses the last which is of the form $\mathsf{inl}(-)$, if any, otherwise returning $\mathsf{inr}(\star)$. Because g' assumes at most one value in A we get that $\tilde{g}$ is monotone, $q : \mathsf{ismon}(\tilde{g})$. Furthermore $(\tilde{g}, q) \downarrow_{\mathsf{Seq}} a$ holds if and only if we have $\Sigma_{n:\mathbb{N}}\, g'(\sigma_n) = \mathsf{inl}(a)$.

In order to complete the proof of $P\left(\bigsqcup(f,p)\right)$ we show that $w(\tilde{g}, q) = \bigsqcup(f,p)$ by using antisymmetry:

– *First part:* $w(\tilde{g}, q) \sqsubseteq \bigsqcup(f,p)$. After unfolding the definition of w we see that it suffices to prove $[\eta \mid \bot]\,(\tilde{g}_n) \sqsubseteq \bigsqcup(f,p)$ for an arbitrary $n : \mathbb{N}$. If $\tilde{g}_n$ is $\mathsf{inr}(\star)$, then this is trivial. If $\tilde{g}_n$ is $\mathsf{inl}(a)$ for some $a : A$, then we can find a pair (i,j) such that $g'_{i,j} = \mathsf{inl}(a)$. Thus we get the following chain:

$$[\eta \mid \bot]\,(\tilde{g}_n) \;=\; [\eta \mid \bot]\,(g'_{i,j}) \;\sqsubseteq\; w(g_i) \;=\; f_i \;\sqsubseteq\; \bigsqcup(f,p)$$

– *Second part:* $\bigsqcup(f,p) \sqsubseteq w(\tilde{g}, q)$. Given $n : \mathbb{N}$, we show that $f_n \sqsubseteq w(\tilde{g}, q)$. By γ_n we have $f_n = w(g_n)$. Thus it suffices to prove $w(g_n) \sqsubseteq w(\tilde{g}, q)$, which by Lemma 12 follows if $g_n \sqsubseteq_{\mathsf{Seq}} (\tilde{g}, q)$. If $g_n(i) = \mathsf{inl}(a)$ for some i and a, then we have $\mathsf{inl}(a) = g'_{n,i} = g'(\sigma(\sigma^{-1}_{n,i}))$, and thus $(\tilde{g}, q) \downarrow_{\mathsf{Seq}} a$. □

This immediately shows that $\tilde{w}$ is surjective as well. Putting the pieces together, we get the main result of this section:

Theorem 15. *In the presence of countable choice the map $\tilde{w}$ is an equivalence. Hence the three sets $\mathrm{D}(A)/{\sim_{\mathrm{D}}}$, $\mathsf{Seq}_A/{\sim_{\mathsf{Seq}}}$ and $A_\bot$ are equivalent.*

Proof. A function between sets is an equivalence exactly if it is surjective and injective. This is a special case of Theorem 4.6.3 in the HoTT book [20], which states that a function between arbitrary types is an equivalence if and only if it is surjective and an *embedding*, which for sets is equivalent to being injective. □

5 Applications

The following examples show that our construction can be used in formalisations.

5.1 Nonterminating Functions as Fixed Points

Partiality algebras can be used to implement not necessarily terminating functions. Let $(Y, \sqsubseteq_Y, \bot_Y, \eta_Y, \bigsqcup_Y)$ be a partiality algebra, and let $\varphi : Y \to Y$ be a monotone and ω-continuous function. We can write down the least fixed point of φ directly as $\bigsqcup_Y(\lambda n.\varphi^n(\bot_Y), p)$, where p is constructed from the fact that $\bot_Y \sqsubseteq_Y \varphi(\bot_Y)$ and from the monotonicity proof of φ. One does not need ω-continuity to write down this expression, but we use it to prove that the expression is a fixed point of φ.

If $(Y, \sqsubseteq_Y, \bot_Y, \eta_Y, \bigsqcup_Y)$ is a partiality algebra and X is any type, then the function space $X \to Y$ can be given the structure of a partiality algebra in a canonical way (this is done for *dependent* types $\Pi_{x:X}\, Y(x)$ in the formalisation).

As an example of how this kind of partiality algebra can be used we will construct a function $search_q : A^\omega \to A_\perp$ that takes an element of the coinductive set of streams A^ω and searches for an element of the set A satisfying the decidable predicate $q : A \to \mathbf{2}$. The function is constructed as the least fixed point of the following endofunction on $A^\omega \to A_\perp$:

$$\Phi(f)(a :: as) :\equiv \mathsf{if}\ q(a)\ \mathsf{then}\ \eta(a)\ \mathsf{else}\ f(as)$$

It is straightforward to check that $f \sqsubseteq g$ implies $\Phi(f) \sqsubseteq \Phi(g)$ by applying $\Phi(f)$ and $\Phi(g)$ to a point $a :: as$ and doing case analysis on $q(a)$. Thus Φ is monotone. In a similar way one can verify that Φ is ω-continuous.

5.2 Functions from the Reals

Let us consider the Cauchy reals, defined as a quotient. We say that $f : \mathbb{N} \to \mathbb{Q}$ is a *Cauchy sequence* if, for all $m, n : \mathbb{N}$ with $m < n$, we have $-1 < m \cdot (f_m - f_n) < 1$. Furthermore, f and g are equivalent (written $f \sim g$) if, for all $n : \mathbb{N}$, we have $-2 \leq n \cdot (f_n - g_n) \leq 2$. We use the notation $\mathbb{R}^q$ for the quotient of Cauchy sequences by $\sim$.

A meta-theoretic result is that, without further assumptions, any definable function (i.e. any closed term) of type $\mathbb{R}^q \to \mathbf{2}$ is constant for reasons of continuity [16]. In particular, we cannot define a function *isPositive* which checks whether a real number is positive. However, we *can* define a function $isPositive : \mathbb{R}^q \to \mathbf{2}_\perp$ such that $isPositive(r)$ is equal—but not judgmentally/definitionally equal—to $\eta(1_\mathbf{2})$ if r is positive, $\eta(0_\mathbf{2})$ if r is negative, and $\perp$ if r is zero.

We define this function as follows: Given a Cauchy sequence $f : \mathbb{N} \to \mathbb{Q}$, we construct a new sequence $\overline{f} : \mathbb{N} \to \{-, ?, +\}$. The idea is that $\overline{f}_n$ is an approximation which only takes f_i with $i \leq n$ into account. We start with $\overline{f}_0 :\equiv ?$. If we have chosen $\overline{f}_{n-1}$ to be $-$, then we choose $\overline{f}_n$ to be $-$ as well, and analogously for $+$. If we have chosen $\overline{f}_{n-1}$ to be $?$, we check whether $f_n \cdot n < -2$, in which case we choose $\overline{f}_n$ to be $-$; if $f_n \cdot n > 2$, we choose $\overline{f}_n$ to be $+$; otherwise, we choose $\overline{f}_n$ to be $?$. We can compose with the map $\{-, ?, +\} \to A_\perp$ which is defined by $- \mapsto \eta(0_\mathbf{2})$, $? \mapsto \perp$, and $+ \mapsto \eta(1_\mathbf{2})$. This defines a monotone sequence in $\mathbf{2}_\perp$, and we can form $\bigsqcup \overline{f} : \mathbf{2}_\perp$ to answer whether f represents a positive or negative number, or is zero. One can check that equivalent Cauchy sequences get mapped to equal values, hence we get $isPositive : \mathbb{R}^q \to \mathbf{2}_\perp$.

The strategy outlined above does not quite work for the reals defined as a HIIT [20] because, roughly speaking, in that setting f_n is a real number and a comparison such as $f_n \cdot n < -2$ is undecidable. Recently Gilbert has refined our approach and defined a function *isPositive* for such reals [13], using the definition of the partiality monad presented in this text (with insignificant differences). Gilbert's key observation is that comparisons between real numbers and rational numbers are semidecidable, and semidecidability is sufficient to define *isPositive*.

5.3 Operational Semantics

In previous work the second-named author has discussed how one can use the delay monad to express operational semantics as definitional interpreters [11]. As a case study we have ported some parts of this work to the partiality monad discussed in the present text: definitional interpreters for a simple functional language and a simple virtual machine, a type soundness proof, a compiler, and a compiler correctness result. Due to lack of space we do not include any details here, but refer interested readers to the accompanying source code.

6 Discussion and Further Work

We have constructed a partiality monad without using countable choice. This is only a first step in the development of a form of constructive domain theory in type theory. It remains to be seen whether it is possible to, for instance, replicate the work of Benton et al. [5], who develop domain theory using the delay monad.

Consider the partial function $\mathit{filter} : \Pi_{A:\mathsf{Set}}\,(A \to \mathbf{2}) \to A^\omega \to A^\omega$ that filters out elements from a stream. How should partial streams over A be defined? Defining them as $\nu X.(A \times X)_\perp$ seems inadequate, because the ordering of $(-)_\perp$ is flat. One approach would perhaps be to define this type by solving a domain equation. Instead of relying on the type-theoretic mechanism to define recursive types, we can perhaps construct a suitable type of partial streams as the colimit of an ω-cocontinuous functor on the category of ω-cpos. Preliminary investigations indicate that QIITs are useful in the endeavour, for example in the definition of a lifting comonad on ω-cpos (as suggested by Paolo Capriotti).

Going in another direction, it might be worth investigating how much topology can be done using the Sierpinski space, represented as $\mathbf{1}_\perp$ in our setting. A very similar question was discussed at the Special Year on Univalent Foundations of Mathematics at the IAS in Princeton (2012–2013). Moreover, some observations have been presented by Gilbert [13], who used our definition of $\mathbf{1}_\perp$ as presented in this paper (with minor differences).

Acknowledgements. We thank Gershom Bazerman, Paolo Capriotti, Bernhard Reus, and Bas Spitters for interesting discussions and pointers to related work, and the anonymous reviewers for useful feedback. The work presented in Sect. 3.3 was done in collaboration with Paolo Capriotti.

References

1. Altenkirch, T., Kaposi, A.: Type theory in type theory using quotient inductive types. In: Proceedings of the 43rd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2016, pp. 18–29 (2016). doi:10.1145/2837614.2837638
2. Altenkirch, T., Capriotti, P., Dijkstra, G., Nordvall Forsberg, F.: Quotient inductive-inductive types. Preprint arXiv:1612.02346v1 [cs.LO] (2016)

3. Altenkirch, T., Danielsson, N.A., Kraus, N.: Code related to the paper "Partiality, revisited: The partiality monad as a quotient inductive-inductive type" (2017). Agda code, http://www.cse.chalmers.se/~nad/
4. Awodey, S., Gambino, N., Sojakova, K.: Inductive types in homotopy type theory. In: 2012 27th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS), pp. 95–104 (2012). doi:10.1109/LICS.2012.21
5. Benton, N., Kennedy, A., Varming, C.: Some domain theory and denotational semantics in coq. In: Berghofer, S., Nipkow, T., Urban, C., Wenzel, M. (eds.) TPHOLs 2009. LNCS, vol. 5674, pp. 115–130. Springer, Heidelberg (2009). doi:10.1007/978-3-642-03359-9_10
6. Capretta, V.: General recursion via coinductive types. Logical Methods Comput. Sci. **1**(2), 1–28 (2005). doi:10.2168/LMCS-1(2:1)2005
7. Capretta, V., Altenkirch, T., Uustalu, T.: Partiality is an effect. Slides for a talk given by Uustalu at the 22nd Meeting of IFIP Working Group 2.8 (2005). http://www.cs.ox.ac.uk/ralf.hinze/WG2.8/22/slides/tarmo.pdf
8. Chapman, J., Uustalu, T., Veltri, N.: Quotienting the delay monad by weak bisimilarity. In: Leucker, M., Rueda, C., Valencia, F.D. (eds.) ICTAC 2015. LNCS, vol. 9399, pp. 110–125. Springer, Cham (2015). doi:10.1007/978-3-319-25150-9_8
9. Cockx, J., Abel, A.: Sprinkles of extensionality for your vanilla type theory. In: TYPES 2016, Types for Proofs and Programs, 22nd Meeting, Book of Abstracts (2016). http://www.types2016.uns.ac.rs/images/abstracts/cockx.pdf
10. Coquand, T., Mannaa, B., Ruch, F.: Stack semantics of type theory. Preprint arXiv:1701.02571v1 [cs.LO] (2017)
11. Danielsson, N.A.: Operational semantics using the partiality monad. In: Proceedings of the 17th ACM SIGPLAN International Conference on Functional Programming, ICFP 2012, pp. 127–138 (2012). doi:10.1145/2364527.2364546
12. Dijkstra, G.: Quotient inductive-inductive definitions. Ph.D. thesis, University of Nottingham (2017). In preparation
13. Gilbert, G.: Formalising real numbers in homotopy type theory. In: Proceedings of the 6th ACM SIGPLAN Conference on Certified Programs and Proofs, CPP 2017, pp. 112–124 (2017). doi:10.1145/3018610.3018614
14. Hofmann, M.: Extensional concepts in intensional type theory. Ph.D. thesis, University of Edinburgh (1995)
15. Kraus, N., Escardó, M., Coquand, T., Altenkirch, T.: Generalizations of Hedberg's theorem. In: Hasegawa, M. (ed.) TLCA 2013. LNCS, vol. 7941, pp. 173–188. Springer, Heidelberg (2013). doi:10.1007/978-3-642-38946-7_14
16. Li, N.: Quotient types in type theory. Ph.D. thesis, University of Nottingham (2015)
17. Lubarsky, R.S.: On the Cauchy completeness of the constructive Cauchy reals. Math. Logic Quart. **53**(4–5), 396–414 (2007). doi:10.1002/malq.200710007
18. Richman, F.: Constructive mathematics without choice. In: Schuster, P., Berger, U., Osswald, H. (eds.) Reuniting the Antipodes – Constructive and Nonstandard Views of the Continuum. Synthese Library, vol. 306, pp. 199–205. Springer Science+Business Media, Dordrecht (2001). doi:10.1007/978-94-015-9757-9_17
19. Sojakova, K.: Higher inductive types as homotopy-initial algebras. In: Proceedings of the 42nd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2015, pp. 31–42 (2015). doi:10.1145/2676726.2676983
20. The Univalent Foundations Program: Homotopy Type Theory: Univalent Foundations of Mathematics, 1st edn. (2013). https://homotopytypetheory.org/book/