The Delay Monad and Restriction Categories

Tarmo Uustalu and Niccolò Veltri(✉)

Department of Software Science, Tallinn University of Technology,
Akadeemia tee 21B, 12618 Tallinn, Estonia
{tarmo,niccolo}@cs.ioc.ee

Abstract. We continue the study of Capretta's delay monad as a means of introducing non-termination from iteration into Martin-Löf type theory. In particular, we explain in what sense this monad provides a canonical solution. We discuss a class of monads that we call ω-complete pointed classifying monads. These are monads whose Kleisli category is an ω-complete pointed restriction category where pure maps are total. All such monads support non-termination from iteration: this is because restriction categories are a general framework for partiality; the presence of an ω-join operation on homsets equips a restriction category with a uniform iteration operator. We show that the delay monad, when quotiented by weak bisimilarity, is the initial ω-complete pointed classifying monad in our type-theoretic setting. This universal property singles it out from among other examples of such monads.

1 Introduction

The delay datatype was introduced by Capretta [4] in order to facilitate the definition of non-terminating functions in type theory and has been used as such by several authors, see, e.g., Danielsson's work [10] for an application to operational semantics or Benton et al.'s work [2] on domain theory in type theory. Inhabitants of the delay datatype are "delayed values", called computations throughout this paper. They can be non-terminating and not return a value at all. Often, one is only interested in termination of computations and not the exact computation time. Identifying computations that only differ by finite amounts of delay corresponds to quotienting the delay datatype by termination-sensitive weak bisimilarity. In earlier work [5], we showed that the monad structure of the delay datatype is preserved under quotienting by weak bisimilarity in an extension of type theory with inductive-like quotient types à la Hofmann [14], proposition extensionality and the axiom of countable choice.

It is common in the type-theoretic programming community to say that the quotiented delay monad is useful for "modeling partial functions" or "introducing non-termination as an effect" in type theory. In this paper, we explain in what sense exactly this monad meets these aims. To do so, we introduce the notion of ω-complete pointed classifying monad. Such a monad is first of all a "monad for partiality", in that its Kleisli category is a restriction category where pure maps are total. Cockett and Lack [8] have termed such monads classifying monads;

D.V. Hung and D. Kapur (Eds.): ICTAC 2017, LNCS 10580, pp. 32–50, 2017.
DOI: 10.1007/978-3-319-67729-3_3

the restriction categories of Cockett and Lack [7] are an axiomatic approach to partiality where every partial function is required to define a partial endofunction on its domain, the corresponding partial identity function, meeting certain equational conditions. Moreover, an ω-complete pointed classifying monad is a "monad for non-termination", in that its Kleisli category is ω**CPPO**-enriched wrt. the "less defined than" order on homsets induced by the restriction operation. In other words, the Kleisli category is an ω-complete pointed restriction category (in a sense that is analogous to finite-join restriction categories [13]).

We show that the quotiented delay datatype possesses an ω-complete pointed classifying monad structure. To this end, we first prove that the quotiented delay datatype delivers free ω-complete pointed partial orders. From this, we further prove that the quotiented delay datatype is the initial ω-complete pointed classifying monad. Intuitively, this tells us that the Kleisli category of this monad is the minimal setting in Martin-Löf type theory for non-terminating functions.

The initiality result is only interesting, if the category of ω-complete pointed classifying monads contains at least some other interesting examples. We show that the datatype of "values on conditions" also possesses an ω-complete pointed classifying monad structure and observe that it is not isomorphic to the quotiented delay monad.

Throughout the paper, we reason constructively, in type theory. The maybe monad is therefore a classifying monad, but not an ω-complete pointed classifying monad. Classically, both the delay monad and the conditional monad are isomorphic to the maybe monad and thus just complications of something that can be expressed much simpler. But constructively they are very different.

The paper is organized as follows. In Sect. 2, we define ω-complete pointed classifying monads and prove some properties about them. In Sect. 3, we introduce the delay monad and weak bisimilarity. In Sect. 4, we quotient the delay monad by weak bisimilarity and we show that the resulting monad is a classifying monad. In Sect. 5, we construct an alternative monad structure on the delay datatype, which makes it an almost-classifying monad. In Sect. 6, we prove that the quotiented delay datatype is the initial ω-complete pointed classifying monad. In Sect. 7, we present some other examples of ω-complete pointed classifying monads. Finally, in Sect. 8, we draw some conclusions and discuss future work.

Our discussion on ω-complete pointed classifying monads applies to general categories. The discussion of the delay monad is carried out for **Set**; generalizing it is future work. We reiterate that we only accept constructive reasoning.

We have fully formalized the development of the paper in the dependently typed programming language Agda [16]. The code is available at http://cs.ioc.ee/~niccolo/omegacpcm/. It uses Agda version 2.4.2.3 and Agda standard library version 0.9.

The Type-Theoretical Framework. Our work is settled in Martin-Löf type theory extended with the following extensional concepts: function extensionality (pointwise equal functions are equal), proposition extensionality (logically equivalent propositions are equal) and inductive-like quotient types à la Hofmann [14].

Equivalently, we could work in homotopy type theory, where function and proposition extensionality are consequences of the univalence axiom and quotient types are definable as higher inductive types. Remember that a type is a proposition when every two of its inhabitants are equal.

We assume uniqueness of identity proofs, which corresponds to working with 0-truncated types in homotopy type theory. We also assume that strongly bisimilar coinductive data are equal.

We write $=$ for definitional equality and $\equiv$ for propositional equality (the identity type).

We review quotient types. Let X be a type and R an equivalence relation on X. Given another type Y, we say that a function $f : X \to Y$ is *R-compatible* (or simply compatible, when the equivalence relation is clear from the context) if $x_1 R x_2$ implies $f\, x_1 \equiv f\, x_2$. The notion of R-compatibility extends straightforwardly to dependent functions. The *quotient* of X by R is described by the following data:

(i) a carrier type X/R;
(ii) a R-compatible map $[\_] : X \to X/R$;
(iii) a dependent eliminator: for every family of types $Y : X/R \to \mathcal{U}_k$ and R-compatible function $f : \prod_{x:X} Y\,[x]$, there exists a function $\mathsf{lift}\, f : \prod_{q:X/R} Y\, q$ such that $\mathsf{lift}\, f\, [x] \equiv f\, x$ for all $x : X$.

We postulate the existence of the above data for all X and R.

2 ω-Complete Pointed Classifying Monads

In this section, we introduce our monads for non-termination. We call them ω-complete pointed classifying monads. Their definition is built on Cockett and Lack's restriction categories and classifying monads [7,8] and Cockett and Guo's finite-join restriction categories [13]. Throughout this section, we work in a fixed base category $\mathbb{C}$.

2.1 Classifying Monads

First, some notation. Given a monad $T = (T, \eta, (-)^*)$, we write $\mathbf{Kl}(T)$ for its Kleisli category. We write $g \diamond f$ for the composition $g^* \circ f$ of g and f in $\mathbf{Kl}(T)$.

Definition 1. We call a monad T an *almost-classifying monad*, if there exists an operation

$$\frac{f : X \to TY}{\overline{f} : X \to TX}$$

called *restriction*, subject to the following conditions:

CM1 $f \diamond \overline{f} \equiv f$, for all $f : X \to TY$
CM2 $\overline{g} \diamond \overline{f} \equiv \overline{f} \diamond \overline{g}$, for all $f : X \to TY$ and $g : X \to TZ$

CM3 $\overline{g} \diamond \overline{f} \equiv \overline{g \diamond \overline{f}}$, for all $f : X \to TY$ and $g : X \to TZ$
CM4 $\overline{g} \diamond f \equiv f \diamond \overline{g \diamond f}$, for all $f : X \to TY$ and $g : Y \to TZ$
CM5 $\overline{\eta_Y \circ f} \equiv \eta_X$, for all $f : X \to Y$.

We call it a *classifying monad*, if it also satisfies

CM6 $\overline{\mathsf{id}_{TX}} \equiv T\eta_X$.

In other words, T is an almost-classifying monad, if its Kleisli category $\mathbf{Kl}(T)$ is a restriction category (conditions CM1–CM4) in which pure maps are total (condition CM5). The restriction of a map $f : X \to TY$ should be thought of as a "partial identity function" on X, a kind of a specification, in the form of a map, of the "domain of definedness" of f (which need not be present in the category as an object). A map $f : X \to TY$ is called *total*, if its restriction is the identity function on X in $\mathbf{Kl}(T)$, i.e., if $\overline{f} \equiv \eta_X$.

The additional condition CM6 of a classifying monad was postulated by Cockett and Lack in order to connect classifying monads and partial map classifiers, or more generally, classified restriction categories and classified $\mathcal{M}$-categories (Theorem 3.6 of [8]), $\mathcal{M}$-categories being Robinson and Rosolini's [17] framework for partiality. While it fulfills this purpose, this condition is very restrictive for other purposes. First of all, it forbids a general monad T from being a classifying monad whose Kleisli category has all maps total. Indeed, define $\overline{f} = \eta_X$, for all $f : X \to TY$. Then conditions CM1–CM5 trivially hold, while condition CM6 is usually false, since generally $\overline{\mathsf{id}_{TX}} \equiv \eta_{TX} \not\equiv T\eta_X$.

Notice that the condition CM1 is a consequence of CM4 and CM5:

$$f \diamond \overline{f} \equiv f \diamond \overline{\eta_Y \diamond f} \overset{\text{CM4}}{\equiv} \overline{\eta_Y} \diamond f \overset{\text{CM5}}{\equiv} \eta_Y \diamond f \equiv f$$

Definition 2. A *classifying monad morphism* between classifying monads T and S, with restrictions $\overline{(-)}$ resp. $\widetilde{(-)}$, is a monad morphism σ between the underlying monads such that $\sigma \circ \overline{f} \equiv \widetilde{\sigma \circ f}$, for all $f : X \to TY$.

(Almost) classifying monads and (almost) classifying monad morphisms form categories.

An important class of classifying monads is given by the equational lifting monads of Bucalo et al. [3]. Recall that a strong monad T, with left strength ψ, is called *commutative*, if the following diagram commutes:

$$\begin{array}{ccc} T\,X \times T\,Y & \xrightarrow{\psi_{T\,X,\,Y}} & T\,(T\,X \times Y) \\ {\scriptstyle \phi_{X,\,TY}}\downarrow & & \downarrow{\scriptstyle \phi^*_{X,\,Y}} \\ T\,(X \times T\,Y) & \xrightarrow{\psi^*_{X,\,Y}} & T\,(X \times Y) \end{array}$$

Here $\phi = T\mathsf{swap} \circ \psi \circ \mathsf{swap}$ is the right strength.

Definition 3. An *equational lifting monad* is a commutative monad making the following diagram commute:

$$\begin{array}{ccc} TX & \xrightarrow{\Delta} & TX \times TX \\ {\scriptstyle T\Delta}\downarrow & & \downarrow{\scriptstyle \psi_{TX,X}} \\ T(X \times X) & \xrightarrow{T(\eta_X \times \mathsf{id}_X)} & T(TX \times X) \end{array} \tag{1}$$

Every equational lifting monad is canonically a classifying monad. Its restriction operation is defined with the aid of the strength:

$$\overline{f} = \; X \xrightarrow{\langle \mathsf{id}_X, f\rangle} X \times TY \xrightarrow{\psi_{X,Y}} T(X \times Y) \xrightarrow{T\mathsf{fst}} TX$$

Notice that, in order to construct an almost-classifying monad, we can relax condition (1) above and consider Cockett and Lack's copy monads [9].

Definition 4. A *copy monad* is a commutative monad making the following diagram commute:

$$\begin{array}{ccc} TX & \xrightarrow{\Delta} & TX \times TX \\ {\scriptstyle T\Delta}\downarrow & & \downarrow{\scriptstyle \psi_{TX,X}} \\ T(X \times X) & \xleftarrow{\phi^*_{X,X}} & T(TX \times X) \end{array}$$

Every equational lifting monad is a copy monad:

$$\phi^* \circ \psi \circ \Delta \equiv \phi^* \circ T\langle \eta, \mathsf{id}\rangle \equiv (\phi \circ (\eta \times \mathsf{id}) \circ \Delta)^* \equiv (\eta \circ \Delta)^* \equiv T\Delta$$

Every copy monad is canonically an almost-classifying monad. Its restriction operation is defined as for an equational lifting monad.

2.2 ω-Joins

The Kleisli category of a classifying monad is equipped with a partial order called the *restriction order*: $f \leq g$ if and only if $f \equiv g \diamond \overline{f}$. That is, f is less defined than g, if f coincides with g on f's domain of definedness. Notice that, for all $f : X \to TY$, we have $\overline{f} \leq \eta_X$.

Lemma 1. *Given a classifying monad T:*

(i) the ordering $\leq$ makes $\mathbf{Kl}(T)$ **Poset***-enriched, i.e., for all $h : W \to TX$, $f, g : X \to TY$ and $k : Y \to TZ$, if $f \leq g$, then $k \diamond f \diamond h \leq k \diamond g \diamond h$;*
(ii) if $f \leq g$, then $\overline{f} \leq \overline{g}$, for all $f, g : X \to TY$.

Given a stream $s : \mathbb{N} \to (X \to TY)$, we say that s is *increasing* (or a *chain*) with respect to $\leq$, and we write $\mathsf{isIncr}_{\leq}\, s$, if $s\, n \leq s\, (n + 1)$, for all $n : \mathbb{N}$.

Definition 5. A classifying monad T is a *ω-complete pointed classifying monad*, if there exist two operations

$$\frac{}{\bot_{X,Y} : X \to TY} \qquad \frac{s : \mathbb{N} \to (X \to TY) \quad \mathsf{isIncr}_{\leq}\, s}{\bigsqcup s : X \to TY}$$

satisfying the following conditions:

BOT1 $\bot_{X,Y} \leq f$, for all $f : X \to TY$
BOT2 $\bot_{Y,Z} \diamond f \equiv \bot_{X,Z}$, for all $f : X \to TY$
LUB1 $s\, n \leq \bigsqcup s$, for all $n : \mathbb{N}$ and increasing $s : \mathbb{N} \to (X \to TY)$
LUB2 if $s\, n \leq t$ for all $n : \mathbb{N}$, then $\bigsqcup s \leq t$, for all $t : X \to TY$ and increasing $s : \mathbb{N} \to (X \to TY)$
LUB3 $\bigsqcup s \diamond f \equiv \bigsqcup(\lambda n.\, s\, n \diamond f)$, for all $f : X \to TY$ and increasing $s : \mathbb{N} \to (Y \to TZ)$.

Conditions BOT1, LUB1 and LUB2 state that every homset in $\mathbf{Kl}(T)$ is a ω-complete pointed partial order, ωcppo for short. Conditions BOT2 and LUB3 state that precomposition in $\mathbf{Kl}(T)$ is strict and continuous. It is actually possible to prove that $\mathbf{Kl}(T)$ is $\omega\mathbf{CPPO}$-enriched. Moreover, the $\bot$ and $\bigsqcup$ operations interact well with restriction, as stated in the following lemma.

Lemma 2. *Let T be an ω-complete pointed classifying monad. Then the following equalities hold:*

BOT3 $\overline{f} \diamond \bot_{X,Y} \equiv \bot_{X,Z}$, *for all* $f : Y \to TZ$
BOTR $\overline{\bot_{X,Y}} \equiv \bot_{X,X}$
LUB4 $f \diamond \bigsqcup s \equiv \bigsqcup(\lambda n.\, f \diamond s\, n)$, *for all* $f : Y \to TZ$ *and increasing* $s : \mathbb{N} \to (X \to TY)$
LUBR $\overline{\bigsqcup s} = \bigsqcup(\lambda n.\, \overline{s\, n})$, *for all increasing* $s : \mathbb{N} \to (X \to TY)$.

Notice that the right-hand sides of LUB3, LUB4 and LUBR are well defined, i.e., the streams that the $\bigsqcup$ operation is applied to are chains, thanks to Lemma 1.

Definition 6. A *ω-complete pointed classifying monad morphism* between ω-complete pointed classifying monads T and S is a classifying monad morphism σ between the underlying classifying monads such that $\sigma \circ \bot \equiv \bot$ and $\sigma \circ \bigsqcup s \equiv \bigsqcup(\lambda n.\, \sigma \circ s\, n)$, for all increasing $s : \mathbb{N} \to (X \to TY)$.

In the definition above, the least upper bound $\bigsqcup(\lambda n.\, \sigma \circ s\, n)$ is well-defined, since postcomposition with a classifying monad morphism is a monotone operation. In other words, for all $f, g : X \to TY$ with $f \leq g$, we have $\sigma \circ f \leq \sigma \circ g$. ω-complete pointed classifying monads and ω-complete pointed classifying monad morphisms form a category.

2.3 Uniform Iteration

If a category is ω**CPPO**-enriched, it has an iteration operator that is uniform with respect to all maps. Given a monad T whose Kleisli category is ω**CPPO**-enriched, this means that we have an operation

$$\frac{f : X \to T(Y + X)}{f^\dagger : X \to TY}$$

satisfying the conditions

ITE1 $f^\dagger \equiv [\eta_Y, f^\dagger] \diamond f$, for all $f : X \to T(Y + X)$
ITE2 $g \diamond f^\dagger \equiv ([T\mathsf{inl} \circ g, T\mathsf{inr} \circ \eta_X] \diamond f)^\dagger$, for all $f : X \to T(Y+X)$ and $g : Y \to TZ$
ITE3 $(T[\mathsf{id}_{Y+X}, \mathsf{inr}] \circ f)^\dagger \equiv (f^\dagger)^\dagger$, for all $f : X \to T((Y + X) + X)$
ITEU if $f \diamond h \equiv [T\mathsf{inl} \circ \eta_Y, T\mathsf{inr} \circ \eta_h] \diamond g$, then $f^\dagger \diamond h \equiv g^\dagger$, for all $f : X \to T(Y+X)$, $g : Z \to T(Y + Z)$ and $h : Z \to TX$.

The standard definition of uniform iteration operator includes the dinaturality axiom. Recently it has been discovered that the latter is derivable from the other laws [11].

Concretely, the operation $(-)^\dagger$ is defined as follows. Let $f : X \to T(Y + X)$. We construct a stream $s : \mathbb{N} \to (X \to TY)$ by

$$s\,0 = \bot_{X,Y} \qquad s\,(n+1) = [\eta_Y, s\,n] \diamond f$$

The stream s is a chain, since the function $\lambda g.\, [\eta_Y, g] \diamond f$ is order-preserving. We define $f^\dagger = \bigsqcup s$. That $(-)^\dagger$ satisfies ITE1 is checked as follows. Clearly, $f^\dagger \leq [\eta_Y, f^\dagger] \diamond f$, since $s\,n \leq [\eta_Y, f^\dagger] \diamond f$, for all $n : \mathbb{N}$. For the converse inequality $[\eta_Y, f^\dagger] \diamond f \leq f^\dagger$, it is enough to notice that $[\eta_Y, \bigsqcup s] \diamond f \equiv \bigsqcup(\lambda n.\, [\eta_Y, s\,n] \diamond f)$ and that $[\eta_Y, s\,n] \diamond f \leq f^\dagger$, for all $n : \mathbb{N}$.

3 The Delay Monad

We now introduce Capretta's delay monad, first the unquotiented version D and then the quotient $\mathsf{D}_\approx$.

From this section onward, we do not work with a general base category, but specifically with **Set** only. As before, we only admit type-theoretical constructive reasoning. We use the words 'set' and 'type' interchangeably.

For a given type X, each element of $\mathsf{D}\,X$ is a possibly non-terminating "computation" that returns a value of X, if and when it terminates. We define $\mathsf{D}\,X$ as a coinductive type by the rules

$$\frac{}{\mathsf{now}\,x : \mathsf{D}\,X} \qquad \frac{c : \mathsf{D}\,X}{\mathsf{later}\,c : \mathsf{D}\,X}$$

(Here and in the following, single rule lines refer to an inductive definition, double rule lines to a coinductive definition.) The non-terminating computation never is corecursively defined as $\mathsf{never} = \mathsf{later}\,\mathsf{never}$.

Propositional equality is not suitable for coinductive types. We need different notions of equality, namely strong and weak bisimilarity. *Strong bisimilarity* is coinductively defined by the rules

$$\frac{}{\mathsf{now}\, x \sim \mathsf{now}\, x} \qquad \frac{c_1 \sim c_2}{\mathsf{later}\, c_1 \sim \mathsf{later}\, c_2}$$

One cannot prove that strongly bisimilar computations are equal in intensional Martin-Löf type theory. Therefore we postulate an inhabitant for $c_1 \sim c_2 \to c_1 \equiv c_2$ for all $c_1, c_2 : \mathsf{D}\, X$.

Weak bisimilarity is defined in terms of *convergence*. The latter is a binary relation between $\mathsf{D}\, X$ and X relating terminating computations to their values. It is inductively defined by the rules

$$\frac{}{\mathsf{now}\, x \downarrow x} \qquad \frac{c \downarrow x}{\mathsf{later}\, c \downarrow x}$$

Two computations are considered *weakly bisimilar*, if they differ by a finite number of applications of the constructor later (from where it follows classically that they either converge to equal values or diverge). Weak bisimilarity is defined coinductively by the rules

$$\frac{c_1 \downarrow x \quad c_2 \downarrow x}{c_1 \approx c_2} \qquad \frac{c_1 \approx c_2}{\mathsf{later}\, c_1 \approx \mathsf{later}\, c_2}$$

The delay datatype D is a monad. The unit η is the constructor now, while the Kleisli extension bind is corecursively defined as follows:

$$\begin{aligned}
&\mathsf{bind} : (X \to \mathsf{D}\, Y) \to \mathsf{D}\, X \to \mathsf{D}\, Y \\
&\mathsf{bind}\, f\, (\mathsf{now}\, x) = f\, x \\
&\mathsf{bind}\, f\, (\mathsf{later}\, c) = \mathsf{later}\, (\mathsf{bind}\, f\, c)
\end{aligned}$$

We denote by $\mathsf{str} : X \times \mathsf{D}\, Y \to \mathsf{D}\, (X \times Y)$ the strength operation of the monad D (which it has uniquely, as any monad on **Set**).

Theorem 1. *The delay datatype* D *is a commutative monad.*

We do not know how to construct a non-trivial almost-classifying monad structure on D. We believe that such a construction is impossible. In fact, notice that D is not an equational lifting monad. Indeed, consider the computation $c = \mathsf{later}\, (\mathsf{now}\, x)$. We have $\mathsf{str}\, (c, c) \not\sim \mathsf{D}\langle \mathsf{now}, \mathsf{id} \rangle\, c$, since $\mathsf{str}\, (c, c) \sim \mathsf{later}\, (\mathsf{now}\, (c, x))$ and $\mathsf{D}\langle \mathsf{now}, \mathsf{id} \rangle\, c \sim \mathsf{later}\, (\mathsf{now}\, (\mathsf{now}\, x, x))$.

In order to obtain an almost-classifying monad, we work with the following modifications of the functor D.

(i) We identify weak bisimilar computations and work with the delay datatype quotiented by weak bisimilarity, defined as $\mathsf{D}_{\approx}\, X = \mathsf{D}\, X/{\approx}$. $\mathsf{D}_{\approx}$ does not inherit the monad structure from D straightforwardly. A monad structure is definable assuming the axiom of countable choice [5]. The quotiented delay monad $\mathsf{D}_{\approx}$ is an equational lifting monad and therefore a classifying monad. We show this in Sect. 4.

(ii) We change the definition of the Kleisli extension. In this way we are able to construct an almost-classifying monad structure on D without the need of quotienting. We show this in Sect. 5.

4 The Quotiented Delay Monad

We know that $\mathsf{D}_\approx$ is a functor, since we can define $\mathsf{D}_\approx f = \mathsf{lift}\,([\_] \circ \mathsf{D}\, f)$. It is easy to show that the function $[\_] \circ \mathsf{D}\, f$ is compatible with $\approx$. Unfortunately, in the type theory we are working in, the functor $\mathsf{D}_\approx$ does not extend to a monad. It is a pointed endofunctor, since we can define the unit as $[\_] \circ \mathsf{now}$. But we are unable to define the multiplication. In order to overcome this obstacle, we assume the *axiom of countable choice*. In our setting, this principle can be formulated as follows: given a type X and an equivalence relation R on it, the following isomorphism holds:

$$\mathbb{N} \to X/R \cong (\mathbb{N} \to X)/(\mathbb{N} \to R)$$

where $f\,(\mathbb{N} \to R)\,g = \prod_{n:\mathbb{N}} (f\,n)\,R\,(g\,n)$. We refer to [5] for details on how to exploit countable choice in order to construct a monad structure on the $\mathsf{D}_\approx$ and for a detailed discussion on why we cannot perform the construction without this additional principle.

Theorem 2. *Assume countable choice. The quotiented delay datatype* $\mathsf{D}_\approx$ *is a monad.*

We call $\eta_\approx$ the unit, $\mathsf{bind}_\approx$ the Kleisli extension and $\mathsf{str}_\approx$ the strength operation of $\mathsf{D}_\approx$.

The monad $\mathsf{D}_\approx$ is commutative, because D is commutative. Moreover, it is an equational lifting monad.

Theorem 3. *Assume countable choice. The monad* $\mathsf{D}_\approx$ *is an equational lifting monad and therefore a classifying monad.*

Proof. We need to prove $\mathsf{str}_\approx\,(q, q) \equiv \mathsf{D}_\approx\langle\eta_\approx, \mathsf{id}\rangle\,q$ for all $q : \mathsf{D}_\approx\,X$. Using the induction principle of quotients, it is sufficient to show that, for all $c : \mathsf{D}\,X$, we have $\mathsf{str}_\approx\,([c], [c]) \equiv \mathsf{D}_\approx\langle\eta_\approx, \mathsf{id}\rangle\,[c]$. Using the computation rule of quotients, we have that $\mathsf{str}_\approx\,([c], [c]) \equiv [\mathsf{str}\,([c], c)]$ and $\mathsf{D}_\approx\langle\eta_\approx, \mathsf{id}\rangle\,[c] \equiv [\mathsf{D}\langle\eta_\approx, \mathsf{id}\rangle\,c]$. Therefore it is sufficient to show $\mathsf{str}\,([c], c) \sim \mathsf{D}\langle\eta_\approx, \mathsf{id}\rangle\,c$ for all $x : \mathsf{D}\;X$. We prove this by corecursion on c:

- if $c = \mathsf{now}\,x$, then both terms are equal to $\mathsf{now}\,([\mathsf{now}\,x], x)$;
- if $c = \mathsf{later}\,c'$, we have to show, after an application of the 2nd constructor of strong bisimilarity, that $\mathsf{str}\,([\mathsf{later}\,c'], c') \sim \mathsf{D}\langle\eta_\approx, \mathsf{id}\rangle\,c'$. This is true since by corecursion we have $\mathsf{str}\,([c'], c') \sim \mathsf{D}\langle\eta_\approx, \mathsf{id}\rangle\,c'$ and we know $[c'] \equiv [\mathsf{later}\,c']$.

□

We continue the construction of an ω-complete pointed classifying monad structure on $\mathsf{D}_{\approx}$ and the proof that it is initial in Sect. 6. In the next section, we show that the datatype D carries a monad structure different from the one presented in Sect. 3. This structure makes D an almost-classifying monad already before quotienting.

5 A Different Monad Structure on D

We show how to endow the type D with a copy monad structure without the need of quotienting by weak bisimilarity. The unit is still now, but we change the Kleisli extension. In order to have an easy description of this construction, it is convenient to give an alternative presentation of the delay datatype. In fact, the type $\mathsf{D}\,X$ is isomorphic to the type of increasing streams over $X + 1$ with respect to the ordering $\leq_{\mathsf{S}}$ on $X + 1$ defined by the rules:

$$\frac{}{\mathsf{inl}\,x \leq_{\mathsf{S}} \mathsf{inl}\,x} \qquad \frac{}{\mathsf{inr}\,* \leq_{\mathsf{S}} \mathsf{inl}\,x}$$

So we define the type $\mathsf{D_S}\,X = \sum_{s:\mathbb{N}\to X+1} \prod_{n:\mathbb{N}} s\,n \leq_{\mathsf{S}} s\,(\mathsf{suc}\,n)$. It is not difficult to show that $\mathsf{D_S}\,X$ is isomorphic to $\mathsf{D}\,X$.

Notice that the stream functor $\mathsf{Stream}\,X = \mathbb{N} \to X$ is a monad. The unit returns a constant stream, while the Kleisli extension on a function $f : X \to \mathsf{Stream}\,Y$ and a stream $s : \mathsf{Stream}\,X$ returns the diagonal of the stream of streams $[f\,(s\,0), f\,(s\,1), f\,(s\,2), \ldots]$. The existence of a distributive law $l_X : (\mathsf{Stream}\,X) + 1 \to \mathsf{Stream}\,(X + 1)$ between the stream monad and the maybe monad induces a monad structure on the functor $\mathsf{Stream}_{+1}\,X = \mathsf{Stream}\,(X + 1)$. Concretely, its unit and Kleisli extension can be described as follows:

$$\begin{aligned}
&\eta_{\mathsf{S}} : X \to \mathsf{Stream}_{+1}\,X\\
&\eta_{\mathsf{S}}\,x\,n = \mathsf{inl}\,x\\
&\\
&\mathsf{bind}_{\mathsf{S}} : (X \to \mathsf{Stream}_{+1}\,Y) \to \mathsf{Stream}_{+1}\,X \to \mathsf{Stream}_{+1}\,Y\\
&\mathsf{bind}_{\mathsf{S}}\,f\,s\,n = \mathsf{case}\ s\,n\ \mathsf{of}\\
&\qquad \mathsf{inl}\,x \mapsto f\,x\,n\\
&\qquad \mathsf{inr}\,* \mapsto \mathsf{inr}\,*
\end{aligned}$$

It is easy to see that $\eta_{\mathsf{S}}\,x$ is increasing wrt. $\leq_{\mathsf{S}}$, for all $x : X$. Moreover, given a function $f : X \to \mathsf{D_S}\,Y$ and an increasing stream $s : \mathsf{Stream}_{+1}\,X$, the stream $\mathsf{bind}_{\mathsf{S}}\,(\mathsf{fst} \circ f)\,s$ is also increasing. Thus, $\mathsf{D_S}$ inherits the monad structure from Stream_{+1}.

Since the types $\mathsf{D_S}\,X$ and $\mathsf{D}\,X$ are isomorphic, we also described a monad structure on D. Intuitively, the new Kleisli extension on D, that we call $\mathsf{bind}_{\wedge}$, acts on a function $f : X \to \mathsf{D}\,Y$ and a computation $c : \mathsf{D}\,X$ as follows: if $c = \mathsf{never}$, then $\mathsf{bind}_{\wedge}\,f\,c = \mathsf{never}$; if $c \downarrow x$, then $\mathsf{bind}_{\wedge}\,f\,c = c \wedge f\,x$, where the operation $\wedge$ is corecursively defined with the help of the auxiliary operation $\wedge'$:

$$
\begin{aligned}
&\wedge' : \mathsf{D}\,X \to \mathsf{D}\,Y \to \mathsf{D}\,(X \times Y)\\
&\mathsf{now}\,x \wedge' \mathsf{now}\,y = \mathsf{now}\,(x, y)\\
&\mathsf{now}\,x \wedge' \mathsf{later}\,c_2 = \mathsf{later}\,(\mathsf{now}\,x \wedge' c_2)\\
&\mathsf{later}\,c_1 \wedge' \mathsf{now}\,y = \mathsf{later}\,(c_1 \wedge' \mathsf{now}\,y)\\
&\mathsf{later}\,c_1 \wedge' \mathsf{later}\,c_2 = \mathsf{later}\,(c_1 \wedge' c_2)
\end{aligned}
$$

$$
\begin{aligned}
&\wedge : \mathsf{D}\,X \to \mathsf{D}\,Y \to \mathsf{D}\,Y\\
&c_1 \wedge c_2 = \mathsf{D}\,\mathsf{snd}\,(c_1 \wedge' c_2)
\end{aligned}
$$

When applied to two computations $\mathsf{later}^k\,(\mathsf{now}\,x)$ and $\mathsf{later}^n\,(\mathsf{now}\,y)$, the operation $\wedge$ returns $\mathsf{later}^{\max(k,n)}\,(\mathsf{now}\,y)$. Notice the difference between $\mathsf{bind}_\wedge$ and the operation bind introduced in Sect. 3. Given $c = \mathsf{later}^k\,(\mathsf{now}\,x)$ and $f\,x = \mathsf{later}^n\,(\mathsf{now}\,y)$, we have:

$$\mathsf{bind}_\wedge\,f\,c = \mathsf{later}^{\max(k,n)}\,(\mathsf{now}\,y) \qquad\qquad \mathsf{bind}\,f\,c = \mathsf{later}^{k+n}\,(\mathsf{now}\,y)$$

After quotienting by weak bisimilarity, the two monad structures on D lift, with the aid of countable choice, to the same monad structure $(\mathsf{D}_\approx, \eta_\approx, \mathsf{bind}_\approx)$.

Theorem 4. *The monad* $(\mathsf{D}, \mathsf{now}, \mathsf{bind}_\wedge)$ *is a copy monad and therefore an almost-classifying monad.*

Proof. We need to prove that, for all $c : \mathsf{D}\,X$, we have $\mathsf{costr}^*_\wedge(\mathsf{str}_\wedge\,(c, c)) \equiv \mathsf{D}\Delta\,c$, where $\mathsf{str}_\wedge$ and $\mathsf{costr}_\wedge$ are the left and right strength operations associated to the monad $(\mathsf{D}, \mathsf{now}, \mathsf{bind}_\wedge)$. It is not difficult to show that the functions $\mathsf{costr}_\wedge \diamond \mathsf{str}_\wedge$ and $\mathsf{D}\Delta$ are both propositionally equal to $\wedge'$. □

6 $\mathsf{D}_\approx$ Is the Initial ω-Complete Pointed Classifying Monad

We move back to the construction of ω-complete pointed classifying monad structure on $\mathsf{D}_\approx$ and initiality. First, we show that $\mathsf{D}_\approx\,X$ is the free ωcppo on X.

6.1 $\mathsf{D}_\approx$ Delivers Free ωcppos

Following [4], we introduce the following relation on $\mathsf{D}\,X$:

$$\frac{c_1 \downarrow x \quad c_2 \downarrow x}{c_1 \sqsubseteq c_2} \qquad \frac{c_1 \sqsubseteq c_2}{\mathsf{later}\,c_1 \sqsubseteq \mathsf{later}\,c_2} \qquad \frac{c_1 \sqsubseteq c_2}{\mathsf{later}\,c_1 \sqsubseteq c_2}$$

The type $c_1 \sqsubseteq c_2$ is inhabited not only if $c_1 \approx c_2$, but also when c_1 has some (possibly infinitely many) laters more than c_2. The relation $\sqsubseteq$ lifts to a relation $\sqsubseteq_\approx$ on $\mathsf{D}_\approx X$, that makes the latter a pointed partial order, with $[\mathsf{never}]$ as least element.

We define a binary operation race on $\mathsf{D}\,X$ that returns the computation with the least number of laters. If two computations c_1 and c_2 converge simultaneously, $\mathsf{race}\,c_1\,c_2$ returns c_1.

$$\begin{aligned}
&\mathsf{race} : \mathsf{D}\,X \to \mathsf{D}\,X \to \mathsf{D}\,X\\
&\mathsf{race}\,(\mathsf{now}\,x)\,c = \mathsf{now}\,x\\
&\mathsf{race}\,(\mathsf{later}\,c)\,(\mathsf{now}\,x) = \mathsf{now}\,x\\
&\mathsf{race}\,(\mathsf{later}\,c_1)\,(\mathsf{later}\,c_2) = \mathsf{later}\,(\mathsf{race}\,c_1\,c_2)
\end{aligned}$$

Notice that generally $\mathsf{race}\,c_1\,c_2$ is not an upper bound of c_1 and c_2, since the two computations may converge to different values. The binary operation race can be extended to an ω-operation $\omega\mathsf{race}$. This operation constructs the first converging element of a chain of computations. It is defined using the auxiliary operation $\omega\mathsf{race}'$:

$$\begin{aligned}
&\omega\mathsf{race}' : (\mathbb{N} \to \mathsf{D}\,X) \to \mathbb{N} \to \mathsf{D}\,X \to \mathsf{D}\,X\\
&\omega\mathsf{race}'\,s\,n\,(\mathsf{now}\,x) = \mathsf{now}\,x\\
&\omega\mathsf{race}'\,s\,n\,(\mathsf{later}\,c) = \mathsf{later}\,(\omega\mathsf{race}'\,s\,(\mathsf{suc}\,n)\,(\mathsf{race}\,c\,(s\,n)))
\end{aligned}$$

The operation $\omega\mathsf{race}'$, when applied to a chain $s : \mathbb{N} \to \mathsf{D}\,X$, a number $n : \mathbb{N}$ and a computation $c : \mathsf{D}\,X$, constructs the first converging element of the chain $s' : \mathbb{N} \to \mathsf{D}\,X$, with $s'\,\mathsf{zero} = c$ and $s'\,(\mathsf{suc}\,k) = s\,(n + k)$. The operation $\omega\mathsf{race}$ is constructed by instantiating $\omega\mathsf{race}'$ with $n = \mathsf{zero}$ and $c = \mathsf{never}$. In this way we have that the first converging element of s is the first converging element of s', since never diverges.

$$\begin{aligned}
&\omega\mathsf{race} : (\mathbb{N} \to \mathsf{D}\,X) \to \mathsf{D}\,X\\
&\omega\mathsf{race}\,s = \omega\mathsf{race}'\,s\,\mathsf{zero}\,\mathsf{never}
\end{aligned}$$

Generally $\omega\mathsf{race}\,s$ is not an upper bound of s. But if s is a chain, then $\omega\mathsf{race}\,s$ is the join of s. The operation $\omega\mathsf{race}$, when restricted to chains, lifts, with the aid of countable choice, to an operation $\omega\mathsf{race}_{\approx}$ on $\mathsf{D}_{\approx}X$, which makes the latter a ωcppo.

Theorem 5. *Assume countable choice. The functor* $\mathsf{D}_{\approx}$ *delivers* ω*cppos.*

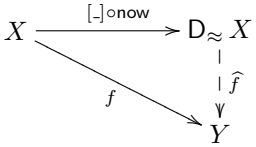
Let $(Y, \leq, \bot, \bigsqcup)$ be an ωcppo and $f : X \to Y$ a function. Every computation over X defines a chain in Y.

$$\begin{aligned}
&\mathsf{cpt2chain}_f : \mathsf{D}\,X \to \mathbb{N} \to Y\\
&\mathsf{cpt2chain}_f(\mathsf{now}\,x)\;n = f\,x\\
&\mathsf{cpt2chain}_f(\mathsf{later}\,c)\;\mathsf{zero} = \bot\\
&\mathsf{cpt2chain}_f(\mathsf{later}\,c)\;(\mathsf{suc}\,n) = \mathsf{cpt2chain}_f\;c\;n
\end{aligned}$$

Given a computation $c = \mathsf{later}^n\,(\mathsf{now}\,x)$ (if $n = \omega$, then $c = \mathsf{never}$), the chain $\mathsf{cpt2chain}_f\,c$ looks as follows:

$$\underbrace{\bot \quad \bot \quad \ldots \quad \bot}_{n} \quad f\,x \quad f\,x \quad f\,x \quad \ldots$$

Therefore it is possible to extend the function f to a function $\widehat{f} : \mathsf{D}\,X \to Y$, $\widehat{f}\,c = \bigsqcup(\mathsf{cpt2chain}_f\,c)$. The function $\widehat{f}$ is $\approx$-compatible, and therefore it can be lifted to a function of type $\mathsf{D}_\approx X \to Y$, that we also name $\widehat{f}$. This function is a ωcppo morphism and it is the unique such morphism making the following diagram commute:



Therefore $\mathsf{D}_\approx X$ is the free ωcppo over X.

Theorem 6. *Assume countable choice. The functor* $\mathsf{D}_\approx$ *delivers free* ω*cppos.*

Recently, Altenkirch et al. [1] constructed a higher inductive-inductive type that delivers free ωcppos by definition without recourse to the axiom of countable choice. To prove that this datatype is isomorphic to the quotiented delay datatype countable choice is again necessary.

6.2 ω-Complete Pointed Classifying Monad Structure on $\mathsf{D}_\approx$ and Initiality

We extend the order $\sqsubseteq_\approx$ to functions in $\mathbf{Kl}(\mathsf{D}_\approx)$ in the usual pointwise way. Let $f, g : X \to \mathsf{D}_\approx Y$, we say that $f \sqsubseteq_\approx g$ if and only if, for all $x : X$, $f\,x \sqsubseteq_\approx g\,x$. (Notice that we use the same notation $\sqsubseteq_\approx$ for functions as well). It is not difficult to show that the order $\sqsubseteq_\approx$ is equivalent to the order associated to the restriction operator that we described in Sect. 2.2.

Lemma 3. *For all* $f, g : X \to \mathsf{D}_\approx Y$*, we have* $f \sqsubseteq_\approx g$ *if and only if* $f \leq g$ *(where* $\leq$ *is the restriction order).*

Theorem 7. *Assume countable choice. The monad* $\mathsf{D}_\approx$ *is an* ω*-complete pointed classifying monad.*

Proof. Let X and Y be two types. The bottom element of the homset $X \to \mathsf{D}_\approx Y$ is the constant map $\lambda\_.\,[\mathsf{never}]$. Let $s : \mathbb{N} \to (X \to \mathsf{D}_\approx Y)$ be a chain wrt. $\leq$. We define

$$\bigsqcup\nolimits_\approx s : X \to \mathsf{D}_\approx Y$$
$$\left(\bigsqcup\nolimits_\approx s\right) x = \omega\mathsf{race}_\approx\,(\lambda n.\, s\,n\,x)$$

where the stream $\lambda n.\, s\,n\,x$ is increasing wrt. $\sqsubseteq_\approx$, which is the case thanks to Lemma 3.

One should now verify that conditions BOT1, BOT2 and and LUB1–LUB3 are met. Conditions BOT1, LUB1 and LUB2 follow directly from $\mathsf{D}_\approx Y$ being a ωcppo, as described in Sect. 6.1. Conditions BOT2 and LUB3 follow from $\mathsf{bind}_\approx$ being a ωcppo morphism between $X \to \mathsf{D}_\approx Y$ and $\mathsf{D}_\approx X \to \mathsf{D}_\approx Y$. □

Let T be an ω-complete pointed almost-classifying monad. We already noted that the type $X \to TY$ is an ωcppo, for all types X and Y. In particular, every type $TX \cong 1 \to TX$ is a ωcppo. Explicitly, given $x_1, x_2 : TX$, we define $x_1 \leq x_2$ as $\lambda *. x_1 \leq \lambda *. x_2$. The bottom element of TX is $\bot_{1,X} *$, while the join of a chain $s : \mathbb{N} \to TX$ is $\bigsqcup(\lambda n. \lambda *. s\, n) *$.

We show that there is a unique ω-complete pointed almost-classifying monad morphism between $\mathsf{D}_\approx$ and T. This characterizes the quotiented delay monad as the universal monad of non-termination.

Theorem 8. *Assume countable choice. $\mathsf{D}_\approx$ is the initial ω-complete pointed almost-classifying monad (and therefore also the initial ω-complete pointed classifying monad).*

Proof. Let $T = (T, \eta, \mathsf{bind})$ be a ω-complete pointed almost-classifying monad. Since TX is a ωcppo and we have a map $\eta_X : X \to TX$, there is a unique ωcppo morphism $\widehat{\eta}$ between $\mathsf{D}_\approx X$ and TX such that $\widehat{\eta} \circ \eta_\approx \equiv \eta$. Therefore, we define

$$\sigma : \mathsf{D}_\approx X \to TX$$
$$\sigma = \widehat{\eta}$$

First, we show that σ is a monad morphism:

- $\sigma \circ \eta_\approx \equiv \eta$ by the universal property of the free ωcppo.
- Given $f : X \to \mathsf{D}_\approx Y$, we have $\sigma \circ \mathsf{bind}_\approx f \equiv \mathsf{bind}\,(\sigma \circ f) \circ \sigma$, because both maps are ωcppo morphisms between $\mathsf{D}_\approx X$ and TY and both maps are equal to $\sigma \circ f$ when precomposed with $\eta_\approx$.

Second, we show that σ is an almost-classifying monad morphism. We have to show that $\sigma \circ \overline{f} \equiv \widetilde{\sigma \circ f}$ for all $f : X \to \mathsf{D}_\approx Y$. Notice that, for all $x : 1 \to X$, we have:

$$\sigma \circ \overline{f} \circ x \overset{\text{CM4}}{\equiv} \sigma \circ \mathsf{D}_\approx x \circ \overline{f \circ x} \overset{\text{nat}}{\equiv} Tx \circ \sigma \circ \overline{f \circ x}$$
$$\widetilde{\sigma \circ f} \circ x \overset{\text{CM4}}{\equiv} Tx \circ \sigma \circ \widetilde{f \circ x}$$

Therefore it is sufficient to show $\sigma \circ \overline{c} \equiv \widetilde{\sigma \circ c}$ for all $c : 1 \to \mathsf{D}_\approx X$. The maps $g\, c = \sigma \circ \overline{c}$ and $h\, c = \widetilde{\sigma \circ c}$ are both strict and continuous maps of type $(1 \to \mathsf{D}_\approx X) \to (1 \to T\, 1)$, and the latter type is isomorphic to $\mathsf{D}_\approx X \to T\, 1$. Notice that since $\mathsf{D}_\approx X$ is the free ωcppo over X, we know that there exists only one strict and continuous map between $\mathsf{D}_\approx X$ and $T\, 1$ that sends terminating computations to $\eta *$. Notice that, for all $x : 1 \to X$, we have

$$g\,(\eta_{\approx X} \circ x) = \sigma \circ \overline{\eta_{\approx X} \circ x} \overset{\text{CM5}}{\equiv} \sigma \circ \eta_{\approx 1} \equiv \eta_1$$
$$h\,(\eta_{\approx X} \circ x) = \sigma \circ \widetilde{\eta_{\approx X}} \circ x \equiv \widetilde{\eta_X \circ x} \overset{\text{CM5}}{\equiv} \eta_1$$

This shows that $g \equiv h$, and therefore σ is a classifying monad morphism.

Finally, σ is a ω-complete pointed almost-classifying monad morphism since $\sigma = \widehat{\eta}$ is a ωcppo morphism between $\mathsf{D}_\approx X$ and TX. In particular, it is strict and continuous.

It remains to check that σ is the unique ω-complete pointed almost-classifying monad morphism between $\mathsf{D}_\approx$ and T. Let τ be another ω-complete pointed almost classifying monad morphism between $\mathsf{D}_\approx$ and T. In particular, for all types X, we have that τ is a ωcppo morphism between $\mathsf{D}_\approx X$ and TX and also $\tau \circ \eta_\approx \equiv \eta$. Therefore, by the universal property of the free ωcppo $\mathsf{D}_\approx$, we have that $\tau \equiv \widehat{\eta} = \sigma$. □

One might wonder whether $\mathbf{Kl}(\mathsf{D}_\approx)$ could be the free ω-complete pointed restriction category over **Set**. This is not the case, since the latter has as objects sets and as maps between X and Y elements of $\mathsf{D}_\approx(X \to Y)$. This observation is an adaptation of a construction by Grandis described by Guo [13].

7 Other Monads of Non-termination

In the previous section, we showed that $\mathsf{D}_\approx$ is the initial ω-complete pointed almost-classifying monad and also the initial ω-complete pointed classifying monad. This would not be a significant result, if the categories of ω-complete pointed classifying and almost-classifying monads were lacking other interesting examples. It is immediate that these categories are non-trivial, since at least the monad $\mathsf{Termin}\, X = 1$ is another ω-complete pointed classifying monad. Since Termin is the final object in the category of monads, it is also the final ω-complete pointed almost-classifying monad and the final ω-complete pointed classifying monad. But of course we are looking for more interesting examples.

7.1 A Non-example: Maybe Monad

The maybe monad $\mathsf{Maybe}\, X = X + 1$ is an example of a classifying monad that is not a ω-complete pointed classifying monad.

The maybe monad is a canonical example of equational lifting monad, so it is a classifying monad. But it is not a ω-complete pointed classifying monad: in order to construct the join of a chain $s : \mathbb{N} \to X + 1$, we need to decide whether there exist an element $x : X$ and a number $k : \mathbb{N}$ such that $s\, k = \mathsf{inl}\, x$, or $s\, n = \mathsf{inr}\, *$ for all $n : \mathbb{N}$. This decision requires the limited principle of omniscience $\mathsf{LPO} = \prod_{s:\mathbb{N}\to 2} (\sum_{n:\mathbb{N}} s\, n \equiv 1) + (\prod_{n:\mathbb{N}} s\, n \equiv 0)$.

7.2 Conditional Monad

For a more interesting example, consider the monad C defined by

$$\mathsf{C}\, X = \sum_{P:\mathcal{U}} \mathsf{isProp}\, P \times (P \to X)$$

where $\mathsf{isProp}\, X = \prod_{x_1,x_2:X} x_1 \equiv x_2$.[1] Intuitively, an element of $\mathsf{C}\, X$ is a proposition P together with an element of X for every proof of P (so at most one

[1] C is typed $\mathcal{U}_1 \to \mathcal{U}_1$, so it is an endofunctor on $\mathbf{Set}_1$. But as the other examples can be replayed for any $\mathcal{U}_k$, comparing this example to them is unproblematic.

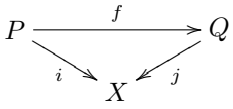
element of X). So a computation produces a condition of its liking and only releases a value of X, if the user can supply a proof; the computation does not give out any hint on how to decide the condition.

The type $\mathsf{C}\,X$ consists of the propositional objects of the slice category of X. The endofunctor C is an equational lifting monad and thus a classifying monad.

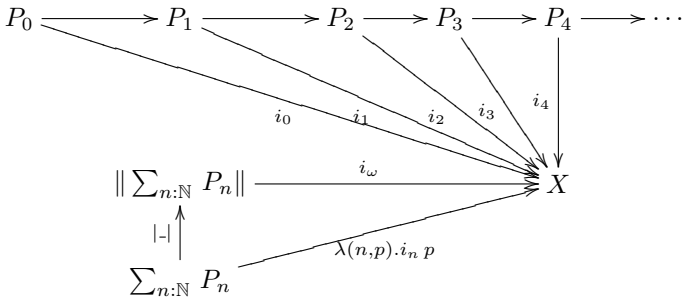
The monad C is a ω-complete pointed classifying monad. This is because every type $\mathsf{C}X$ is a ωcppo. To see this, we first define a partial order on $\mathsf{C}\,X$:

$$(P, ip, i) \sqsubseteq (Q, iq, j) = \sum_{f:P\to Q} \prod_{p:P} i\,p \equiv j\,(f\,p)$$

That is, two elements in $\mathsf{C}X$ are related by $\sqsubseteq$, if there exists a morphism in the slice category of X connecting them:



In other words, the poset $(\mathsf{C}X, \sqsubseteq)$ is the full subcategory of the slice category of X in which objects are propositions. Interestingly, this poset is also a ωcppo. The bottom element is the proposition 0 together with the empty function $0 \to X$. Joins of chains are computed as the following colimit:



where $\| \sum_{n:\mathbb{N}} P_n \|$ is the propositional truncation of $\sum_{n:\mathbb{N}} P_n$, i.e., the quotient of $\sum_{n:\mathbb{N}} P_n$ by the total equivalence relation (which relates every pair of elements in the type). The function i_ω is obtained as the lifting of $\lambda(n,p).i_n\,p : \sum_{n:\mathbb{N}} P_n \to X$. Notice that the latter function is constant, i.e., $i_n\,p_n \equiv i_m\,p_m$ for all $p_n : P_n$ and $p_m : P_m$. In fact, suppose w.l.o.g. $m \leq n$. Then, since the stream is increasing, there exists a function $f : P_m \to P_n$ such that $i_m\,p_m \equiv i_n\,(f\,p_m)$. The type P_n is a proposition, therefore $f\,p_m \equiv p_n$ and $i_n\,p_n \equiv i_m\,p_m$.

As usual, the order $\sqsubseteq$ extends to function spaces. It is not difficult to show that this is equivalent to the restriction order and that C satisfies the laws of an ω-complete pointed classifying monad.

The types $\mathsf{C}\,X$ and $\mathsf{Maybe}\,X$ are isomorphic if and only if the principle of excluded middle for propositions $\mathsf{LEM}_{\mathsf{Prop}} = \prod_{X:\mathcal{U}} \mathsf{isProp}\,X \to X + \neg X$ holds. Since $\mathsf{D}_\approx X$ and $\mathsf{Maybe}\,X$ are isomorphic if and only if LPO holds, and since $\mathsf{LEM}_{\mathsf{Prop}}$ is strictly stronger than LPO, $\mathsf{C}\,X$ is generally not isomorphic to $\mathsf{D}_\approx X$.

The conditional monad C is an instance of a partial map classifier in the sense of [15]. In type theory, partial map classifiers are monads of the form $TX = \sum_{x:D} (x \equiv \top \to X)$ where D is a dominance in the sense of [18] and $\top : D$ is the truth value corresponding to truthfulness. The conditional monad is the partial map classifier associated to the dominance $D = \sum_{X:\mathcal{U}} \mathsf{isProp}\, X$ with $\top = (1, p)$ where p is the simple proof of 1 being a proposition.

7.3 Countable Powerset Monad

An example of an ω-complete pointed almost-classifying monad that is not a classifying monad (since the condition CM6 is not met), is given by the countable powerset construction. This monad is typically employed to model nondeterministic computations. In type theory, the countable powerset monad can be introduced as follows:

$$\mathcal{P}_\infty X = \mathsf{Stream}\,(X + 1)/\mathsf{SameElem}$$

where $\mathsf{SameElem}\, s_1\, s_2 = \prod_{x:X} x \in s_1 \leftrightarrow x \in s_2$ and $x \in s = \sum_{n:\mathbb{N}} s\ n \equiv \mathsf{inl}\, x$. It can be proved that the functor $\mathcal{P}_\infty$ is a monad. This requires the assumption of the axiom of countable choice, because it is defined as a quotient.

Intuitively, the restriction $\overline{f}$ of a map $f : X \to \mathcal{P}_\infty Y$ is the map that, given $x : X$, returns the singleton $\{x\}$, if $f\, x$ is non-empty, and the empty set otherwise. The restriction order on $\mathcal{P}_\infty X \cong 1 \to \mathcal{P}_\infty X$ is thus different from set inclusion. In fact, for $s, t : \mathcal{P}_\infty X$, intuitively $s \leq t$ if and only if $s \equiv t$ or $s \equiv \emptyset$.

7.4 State Monad Transformer

New ω-complete pointed almost-classifying monads can be constructed from already constructed ones with the state monad transformer. Recall that the state monad is defined as $\mathsf{State}\, X = S \to X \times S$, where S is a fixed set of states. Given an ω-complete pointed almost-classifying monad T, the functor $\mathsf{StateT}\, T$ defined by $\mathsf{StateT}\, T\, X = S \to T(X \times S)$ is another ω-complete pointed almost-classifying monad. All operations of $\mathsf{StateT}\, T$ are defined in terms of the operations of T. For example, restriction is constructed as follows:

$$\widetilde{(\_)} : (X \to S \to T(Y \times S)) \to X \to S \to T(X \times S)$$
$$\widetilde{f} = \mathsf{curry}\,(\overline{\mathsf{uncurry}\, f})$$

8 Conclusions

In this paper, we introduced the notion of ω-complete pointed classifying monad. We argued that it explains the idea of "non-termination as an effect". We showed that Capretta's delay monad quotiented by weak bisimilarity is the initial ω-complete pointed classifying monad on **Set** under constructive reasoning and in this sense is the minimal monad for non-termination. We also showed that

the class of ω-complete pointed classifying monads is non-trivial, since it also contains the monad C, which is non-isomorphic to the quotiented delay monad.

ω-complete pointed classifying monads are derived from Cockett and Lack's restriction categories. There are two reasons behind this choice over other category-theoretical approaches to partiality such as partial map categories. Restriction categories, being an axiomatic framework for partiality, are conveniently formalizable in a proof assistant like Agda. This is not the case for partial map categories, whose formalization quickly becomes very involved. Moreover, Cockett and Guo [6] proved that every finite-join restriction category is a full subcategory of the partial map category of an adhesive $\mathcal{M}$-category whose gaps are in $\mathcal{M}$. Therefore, one can come up with a complementary notion of a finite-join classifying monad for partial map categories, but this will inevitably be more involved than the simple notion considered here.

As future work, We would like to generalize this work from **Set** to a general base category while still only accepting constructive reasoning. This requires, first of all, generalizing the definition of the delay monad suitably for less structured categories. Also, we would like to understand whether the delay monad could be the initial completely Elgot monad on **Set** constructively under reasonable semi-classical principles (a monad is said to be a completely Elgot monad, if its Kleisli category has an iteration operator uniform for pure maps). Goncharov et al. [12] have proved that the maybe monad is the initial completely Elgot monad on **Set** classically, but the constructive content of this proof has so far remained elusive for us.

Acknowledgements. This research was supported by the Estonian Ministry of Education and Research institutional research grant IUT33-13 and the Estonian Research Council personal research grant PUT763.

References

1. Altenkirch, T., Danielsson, N.A., Kraus, N.: Partiality, revisited: the partiality monad as a quotient inductive-inductive type. In: Esparza, J., Murawski, A.S. (eds.) FoSSaCS 2017. LNCS, vol. 10203, pp. 534–549. Springer, Heidelberg (2017). doi:10.1007/978-3-662-54458-7_31
2. 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
3. Bucalo, A., Führmann, C., Simpson, A.: An equational notion of lifting monad. Theor. Comput. Sci. **294**(1–2), 31–60 (2003)
4. Capretta, V.: General recursion via coinductive types. Log. Methods Comput. Sci. **1**(2), article 1 (2005)
5. Chapman, J., Uustalu, T., Veltri, N.: Quotienting the delay monad by weak bisimilarity. Math. Struct. Comput. Sci. (to appear)
6. Cockett, J.R.B., Guo, X.: Join restriction categories and the importance of being adhesive. Abstract of talk presented at CT 2007 (2007)

7. Cockett, J.R.B., Lack, S.: Restriction categories I: categories of partial maps. Theor. Comput. Sci. **270**(1–2), 223–259 (2002)
8. Cockett, J.R.B., Lack, S.: Restriction categories II: partial map classification. Theor. Comput. Sci. **294**(1–2), 61–102 (2003)
9. Cockett, J.R.B., Lack, S.: Restriction categories III: colimits, partial limits, and extensivity. Math. Struct. Comput. Sci. **17**(4), 775–817 (2007)
10. Danielsson, N.A.: Operational semantics using the partiality monad. In: Proceedings of 17th ACM SIGPLAN International Conference on Functional Programming, ICFP 2012, pp. 127–138. ACM, New York (2012)
11. Ésik, Z., Goncharov, S.: Some remarks on Conway and iteration theories. arXiv preprint arXiv:1603.00838 (2016)
12. Goncharov, S., Rauch, C., Schröder, L.: Unguarded recursion on coinductive resumptions. Electron. Notes Theor. Comput. Sci. **319**, 183–198 (2015)
13. Guo, X.: Products, joins, meets, and ranges in restriction categories. Ph.D. thesis, University of Calgary (2012)
14. Hofmann, M.: Extensional Constructs in Intensional Type Theory. CPHS/BCS Distinguished Dissertations. Springer, London (1997). doi:10.1007/978-1-4471-0963-1
15. Mulry, P.S.: Partial map classifiers and partial cartesian closed categories. Theor. Comput. Sci. **136**(1), 109–123 (1994)
16. Norell, U.: Dependently typed programming in Agda. In: Koopman, P., Plasmeijer, R., Swierstra, D. (eds.) AFP 2008. LNCS, vol. 5832, pp. 230–266. Springer, Heidelberg (2009). doi:10.1007/978-3-642-04652-0_5
17. Robinson, E., Rosolini, G.: Categories of partial maps. Inf. Comput. **79**(2), 95–130 (1988)
18. Rosolini, G.: Continuity and effectiveness in topoi. DPhil. thesis, University of Oxford (1986)