Math. Struct. in Comp. Science (2019), *vol.* 29, *pp.* 67–92. © Cambridge University Press 2017
doi:10.1017/S0960129517000184 First published online 17 October 2017

Quotienting the delay monad by weak bisimilarity

JAMES CHAPMAN[†], TARMO UUSTALU[‡] and NICCOLÒ VELTRI[‡]

[†]*Department of Computer and Information Sciences, University of Strathclyde, 26 Richmond Street, Glasgow G1 1XH, U.K.*
Email: james.chapman@strath.ac.uk
[‡]*Department of Software Science, Tallinn University of Technology, Akadeemia tee 21B, 12618 Tallinn, Estonia*
Email: tarmo@cs.ioc.ee, niccolo@cs.ioc.ee

Received 5 April 2016; revised 7 July 2017

The delay datatype was introduced by Capretta (*Logical Methods in Computer Science*, 1(2), article 1, 2005) as a means to deal with partial functions (as in computability theory) in Martin-Löf type theory. The delay datatype is a monad. It is often desirable to consider two delayed computations equal, if they terminate with equal values, whenever one of them terminates. The equivalence relation underlying this identification is called weak bisimilarity. In type theory, one commonly replaces quotients with setoids. In this approach, the delay datatype quotiented by weak bisimilarity is still a monad–a constructive alternative to the maybe monad. In this paper, we consider the alternative approach of Hofmann (*Extensional Constructs in Intensional Type Theory*, Springer, London, 1997) of extending type theory with inductive-like quotient types. In this setting, it is difficult to define the intended monad multiplication for the quotiented datatype. We give a solution where we postulate some principles, crucially proposition extensionality and the (semi-classical) axiom of countable choice. With the aid of these principles, we also prove that the quotiented delay datatype delivers free ω-complete pointed partial orders (ωcppos).
Altenkirch et al. (Lecture Notes in Computer Science, vol. 10203, Springer, Heidelberg, 534–549, 2017) demonstrated that, in homotopy type theory, a certain higher inductive–inductive type is the free ωcppo on a type X essentially by definition; this allowed them to obtain a monad of free ωcppos without recourse to a choice principle. We notice that, by a similar construction, a simpler ordinary higher inductive type gives the free countably complete join semilattice on the unit type 1. This type suffices for constructing a monad, which is isomorphic to the one of Altenkirch et al. We have fully formalized our results in the Agda dependently typed programming language.

1. Introduction

The delay datatype was introduced by Capretta (2005) as a means to deal with partial functions (as in computability theory) in Martin-Löf type theory. It is used in this setting to cope with possible non-termination of computations (as, e.g. in the unbounded search of minimalization). Inhabitants of the delay datatype are delayed values that we call computations throughout this paper. Crucially computations can be non-terminating and not return a value at all. The delay datatype constitutes a (strong) monad, which makes it possible to deal with possibly non-terminating computations just like any other flavour of effectful computations following the general monad-based method of Moggi (1991).

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 weak bisimilarity. The quotient datatype is used as a constructive alternative to the maybe datatype, see, e.g. Benton et al. (2009).

Martin-Löf type theory does not have built-in quotient types. The most common approach to compensate for this is to mimic them by working with setoids. But this approach has some troubling shortcomings as well. For example, the concept of a function type is changed (every function has to come with a compatibility proof), the same applies to product types and every other type former. In general, working with setoids in a formal development tends to lead to a lot of largely artificial bureaucracy (sometimes referred to as the 'setoid hell'). An alternative approach, which we pursue here, consists in extending the theory by postulating the existence of inductive-like quotient types à la Hofmann (1997). These quotient types are ordinary types rather than setoids.

In this paper, we ask the question: is the monad structure of the delay datatype preserved under quotienting by weak bisimilarity? Intuitively, this ought to be the case. In the setoid approach, this works out unproblematically indeed. But with inductive-like quotient types, one meets a difficulty when attempting to reproduce the monad structure on the quotiented datatype. Specifically, one cannot define the multiplication. The difficulty has to do with the interplay of the coinductive nature of the delay datatype, or more precisely the infinity involved, and quotient types. We discuss the general phenomenon behind this issue and provide a solution where we postulate some principles, the crucial ones being proposition extensionality (accepted in particular in homotopy type theory) and the (semi-classical) axiom of countable choice. It is very important here to be careful and not postulate too much: in the presence of proposition extensionality, the full axiom of choice implies the law of excluded middle.

We also look at the (strong) arrow structure (in the sense of Hughes (2000)) on the Kleisli homsets for the delay datatype and ask whether this survives quotienting by pointwise weak bisimilarity. Curiously, here the answer is unconditionally positive also for inductive-like quotient types.

Afterwards, to argue that the need for the axiom of countable choice to define the multiplication of the quotiented delay monad is not incidental, but points to an intrinsic issue, we show an additional construction related to the quotiented delay datatype that also requires the same assumption. We show that the quotiented delay datatype applied to a type X is the free ω-complete pointed partial order (ωcppo) over X, under the assumption of countable choice. This construction relates our work to the new work of Altenkirch et al. (2017) in the setting of homotopy type theory. They mimicked the definition of Cauchy reals of the HoTT book (Univalent Foundations Program 2013, Section 11.3) and defined the free ωcppo on X as a higher inductive–inductive type whose constructors build an ωcppo on X and the elimination principle ensures freeness. They also showed that this construction gives a monad (called the partiality monad by the authors); countable choice was not needed to define the monad structure. Under the assumption of countable choice, their monad is isomorphic to ours.

Finally, we construct an alternative datatype for partiality in homotopy type theory. We define it as a partial map classifier (Mulry 1994), classifying partial functions with a semidecidable domain of definedness. Our construction utilizes ordinary higher inductive types rather than higher inductive–inductive types. Specifically, we define the Sierpinski set as a higher inductive type. The resulting monad is isomorphic to the one by Altenkirch et al. and under the assumption of countable choice also to the quotiented delay datatype.

This paper is organized as follows. In Section 2, we give an overview of the type theory we are working in. In Section 3, we introduce the delay datatype and weak bisimilarity. In Section 4, we extend type theory with quotients à la Hofmann. In Section 5, we analyze why a multiplication for the quotiented delay type is impossible to define. We notice that the problem is of a more general nature, and a larger class of types, namely non-wellfounded and non-finitely branching trees, suffers from it. In Section 6, we introduce the axiom of countable choice and derive some important consequences from postulating it. In Section 7, using the results of Section 6, we define multiplication for the delay type quotiented by weak bisimilarity. (We omit the proof of the monad laws, which is the easy part–essentially the proofs for the unquotiented delay datatype carry over.) In Section 8, we quotient the arrow corresponding to the delay monad by pointwise weak bisimilarity. In Section 9, we demonstrate that the quotiented delay datatype delivers free ωcppos, assuming countable choice. In Section 10, we present a new monad for partiality in homotopy type theory and show how it relates to Altenkirch et al.'s construction and to the quotiented delay monad. Finally, in Section 11, we draw some conclusions and discuss future work.

We have fully formalized the results of this paper in the dependently typed programming language Agda (Norell 2009). The formalization is available at `http://cs.ioc.ee/~niccolo/delay/`.

This article is an extended version of an ICTAC 2015 conference paper. Compared to the conference paper, the material of Sections 9 and 10 is entirely new.

2. The type theory under consideration

We consider Martin-Löf type theory with inductive and coinductive types and a cumulative hierarchy of universes $\mathcal{U}_k$. To define functions from inductive types or to coinductive types, we use guarded (co)recursion. We define inductive types by rules with single rule lines and coinductive types by rules with double rule lines. The first universe is simply denoted $\mathcal{U}$, and when we write statements like 'X is a type,' we mean $X : \mathcal{U}$ unless otherwise specified. We allow dependent functions to have implicit arguments and indicate implicit argument positions with curly brackets (as in Agda). We write $\equiv$ for propositional equality (identity types) and $=$ for judgmental (definitional) equality. Reflexivity, transitivity and substitutivity of $\equiv$ are named refl, trans and subst, respectively.

We assume the principle of *function extensionality*, expressing that pointwise equal functions are equal, i.e. the inhabitedness of

$$\mathsf{FunExt} = \prod_{\{X,Y:\mathcal{U}\}} \prod_{\{f_1,f_2:X\to Y\}} \left(\prod_{x:X} f_1\,x \equiv f_2\,x \right) \to f_1 \equiv f_2.$$

Likewise, we will assume analogous extensionality principles stating that strongly bisimilar coinductive data and proofs are equal for the relevant coinductive types and predicates, namely, the delay datatype and weak bisimilarity (check DExt, $\approx\mathsf{Ext}$ below in Sections 3 and 4).

We also assume *uniqueness of identity proofs* for all types, i.e. an inhabitant for

$$\mathsf{UIP} = \prod_{\{X:\mathcal{U}\}} \prod_{\{x_1,x_2:X\}} \prod_{p_1,p_2:x_1 \equiv x_2} p_1 \equiv p_2.$$

A type X is said to be a *proposition*, if it has at most one inhabitant, i.e. if the type

$$\mathsf{isProp}\, X = \prod_{x_1,x_2:X} x_1 \equiv x_2$$

is inhabited.

For propositions, we postulate a further and less standard principle of *proposition extensionality*, stating that logically equivalent propositions are equal:

$$\mathsf{PropExt} = \prod_{\{X,Y:\mathcal{U}\}} \mathsf{isProp}\, X \to \mathsf{isProp}\, Y \to X \leftrightarrow Y \to X \equiv Y.$$

Here, $X \leftrightarrow Y = (X \to Y) \times (Y \to X)$.

Alternatively, we could set our development in *homotopy type theory* (Univalent Foundations Program 2013), but restrict ourselves to work with 0-truncated types, i.e., sets. In the latter framework, the principles FunExt and $\mathsf{PropExt}$ are consequences of the univalence axiom, while the restriction to 0-truncated types implies UIP.

3. Delay monad

For a given type X, each element of the *delay type* $\mathsf{D}\, X$ is a possibly infinite computation that returns a value of X, if it terminates. We define $\mathsf{D}\, X$ as a coinductive type by the rules

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

Let R be an equivalence relation on a type X. The relation lifts to an equivalence relation $\sim_R$ on $\mathsf{D}\, X$ that we call *strong R-bisimilarity*. The relation is coinductively defined by the rules

$$\frac{p : x_1 R\, x_2}{\mathsf{now}_\sim\, p : \mathsf{now}\, x_1 \sim_R \mathsf{now}\, x_2} \qquad \frac{p : c_1 \sim_R c_2}{\mathsf{later}_\sim\, p : \mathsf{later}\, c_1 \sim_R \mathsf{later}\, c_2}$$

We alternatively denote the relation $\sim_R$ with $\mathsf{D}\, R$, since strong R-bisimilarity is the functorial lifting of the relation R to $\mathsf{D}\, X$. Strong $\equiv$-bisimilarity is simply called strong bisimilarity and denoted $\sim$. While it ought to be the case intuitively, one cannot prove that strongly bisimilar computations are equal in Martin-Löf's type theory. Therefore, we postulate an inhabitant for

$$\mathsf{DExt} = \prod_{\{X:\mathcal{U}\}} \prod_{\{c_1,c_2:\mathsf{D}\, X\}} c_1 \sim c_2 \to c_1 \equiv c_2.$$

We take into account another equivalence relation $\approx_R$ on $\mathsf{D}\, X$ called *weak R-bisimilarity*, which is in turn 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

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

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

$$\frac{p_1 : c_1 \downarrow x_1 \quad p_2 : x_1 R\, x_2 \quad p_3 : c_2 \downarrow x_2}{\downarrow_{\approx}\, p_1\, p_2\, p_3 : c_1 \approx_R c_2} \qquad \frac{p : c_1 \approx_R c_2}{\mathsf{later}_{\approx}\, p : \mathsf{later}\,c_1 \approx_R \mathsf{later}\,c_2}$$

Weak $\equiv$-bisimilarity is called just weak bisimilarity and denoted $\approx$. In this case, we modify the first constructor for simplicity:

$$\frac{p_1 : c_1 \downarrow x \quad p_2 : c_2 \downarrow x}{\downarrow_{\approx}\, p_1\, p_2 : c_1 \approx c_2}$$

Remark 3.1. Notice that the type $c_1 \approx c_2$ is not a proposition. But weak bisimilarity can be defined alternatively as the following propositional relation:

$$\frac{}{\mathsf{now}\,x \approx' \mathsf{now}\,x} \qquad \frac{c \downarrow x}{\mathsf{later}\,c \approx' \mathsf{now}\,x} \qquad \frac{c \downarrow x}{\mathsf{now}\,x \approx' \mathsf{later}\,c} \qquad \frac{c_1 \approx' c_2}{\mathsf{later}\,c_1 \approx' \mathsf{later}\,c_2}$$

We have $c \approx' c'$ if and only if $c \approx c'$. We prefer to work with $\approx$ instead of $\approx'$, since proofs of $\approx$ are somewhat easier to construct, there is more freedom. In fact, there are even more robust versions of weak bisimilarity with even more proofs.

The delay datatype D is a (strong) monad. The unit η is the constructor now while the multiplication μ is 'concatenation' of two layers of laters:

$$\begin{aligned}
&\mu : \mathsf{D}\,(\mathsf{D}\,X) \to \mathsf{D}\,X\\
&\mu\,(\mathsf{now}\,c) = c\\
&\mu\,(\mathsf{later}\,c) = \mathsf{later}\,(\mu\,c).
\end{aligned}$$

In the quotients-as-setoids approach, it is trivial to define the corresponding (strong) monad structure on the quotient of D by $\approx$. The role of the quotiented datatype is played by the setoid functor $\widehat{\mathsf{D}}$, defined by $\widehat{\mathsf{D}}\,(X,R) = (\mathsf{D}\,X, \approx_R)$. The unit $\widehat{\eta}$ and multiplication $\widehat{\mu}$ are just η and μ together with proofs of that the appropriate equivalences are preserved. The unit $\widehat{\eta}$ is a setoid morphism from (X,R) to $(D\,X, \approx_R)$, as $x_1\,R\,x_2 \to \mathsf{now}\,x_1 \approx_R \mathsf{now}\,x_2$ by definition of $\approx_R$. The multiplication $\widehat{\mu}$ is a setoid morphism from $(\mathsf{D}\,(\mathsf{D}\,X), \approx_{\approx_R})$ to $(\mathsf{D}\,X, \approx_R)$, since $c_1 \approx_{\approx_R} c_2 \to \mu\,c_1 \approx_R \mu\,c_2$ for all $c_1, c_2 : \mathsf{D}\,(\mathsf{D}\,X)$. The monad laws hold up to $\approx_R$, since they hold up to $\sim_R$.

In this paper, our goal is to establish that the delay datatype quotiented by weak bisimilarity is a monad also in the setting of Hofmann (1997), where the quotient type of a given type has its propositional equality given by the equivalence relation. We discuss such quotient types in the next section.

4. Inductive-like quotients

In this section, we describe quotient types as the particular inductive-like types introduced by M. Hofmann in his PhD thesis (1997). Let X be a type and R an equivalence relation on X. For any type Y and function $f : X \to Y$, we say that f is *R-compatible* (or simply *compatible*, when the intended equivalence relation is clear from the context), if the type

$$\mathsf{compat}\, f = \prod_{\{x_1,x_2:X\}} x_1 R x_2 \to f\,x_1 \equiv f\,x_2$$

is inhabited. The quotient of X by the relation R is described by the following data:

i. a carrier type X/R;
ii. a constructor $[\_] : X \to X/R$ together with a proof $\mathsf{sound} : \mathsf{compat}\,[\_]$;
iii. a dependent eliminator: for every family of types $Y : X/R \to \mathcal{U}_k$ and for every function $f : \prod_{x:X} Y\,[x]$ with $p : \mathsf{dcompat}\, f$, there exists a function $\mathsf{lift}\, f\, p : \prod_{q:X/R} Y\,q$ together with a computation rule

$$\mathsf{lift}_\beta\, f\, p\, x : \mathsf{lift}\, f\, p\, [x] \equiv f\, x$$

for all $x : X$.

The predicate $\mathsf{dcompat}$ is compatibility for dependent functions $f : \prod_{x:X} Y\,[x]$:

$$\mathsf{dcompat}\, f = \prod_{\{x_1,x_2:X\}} \prod_{r:x_1 R x_2} \mathsf{subst}\, Y\, (\mathsf{sound}\, r)\, (f\, x_1) \equiv f\, x_2.$$

We postulate the existence of data (i)–(iii) for all types X and equivalence relations R on X. Notice that the predicate $\mathsf{dcompat}$ depends of the availability of sound. Also notice that, in (iii), we allow elimination on every universe $\mathcal{U}_k$. In our development, we actually eliminate only on $\mathcal{U}$ and once on $\mathcal{U}_1$ (Proposition 4.2).

The *propositional truncation* (or *squash*) $\|X\|$ of a type X is the quotient of X by the total relation $\lambda x_1\, x_2. 1$. We write $|\_|$ instead of $[\_]$ for the constructor of $\|X\|$. The non-dependent version of the elimination principle of $\|X\|$ is employed several times in this paper, so we spell it out: in order to construct a function of type $\|X\| \to Y$, one has to construct a constant function of type $X \to Y$. The type $\|X\|$ can have at most one inhabitant, informally, an 'uninformative' proof of X. For example, an inhabitant of $\|\sum_{x:X} P\,x\|$ can be thought of as a proof of there existing an element of X that satisfies P from which all information has been removed: both the witness element and the proof that it is good. Propositional truncation and other notions of weak or anonymous existence have been thoroughly studied in type theory (Kraus et al. 2013).

We call a function $f : X \to Y$ *surjective*, if the type $\prod_{y:Y} \|\sum_{x:X} f\,x \equiv y\|$ is inhabited, and a *split epimorphism*, if the type $\|\sum_{g:Y\to X} \prod_{y:Y} f\,(g\,y) \equiv y\|$ is inhabited. We say that f is a *retraction*, if the type $\sum_{g:Y\to X} \prod_{y:Y} f\,(g\,y) \equiv y$ is inhabited. Every retraction is a split epimorphism, and every split epimorphism is surjective.

Proposition 4.1. The constructor $[\_]$ is surjective for all quotients.

Proof. Given a type X and an equivalence relation R on X, we define:

$$[\_]\mathsf{surj} : \prod_{q:X/R} \left\| \sum_{x:X} [x] \equiv q \right\|$$
$$[\_]\mathsf{surj} = \mathsf{lift}\,(\lambda\, x.\, |x, \mathsf{refl}|)\, p.$$

The compatibility proof p is trivial, since $|x_1, \mathsf{refl}| \equiv |x_2, \mathsf{refl}|$ for all $x_1, x_2 : X$. □

A quotient X/R is said to be *effective*, if the type $\prod_{x_1,x_2:X} [x_1] \equiv [x_2] \to x_1\, R\, x_2$ is inhabited. In general, effectiveness does not hold for all quotients. But we can prove that all quotients satisfy a weaker property. We say that a quotient X/R is *weakly effective*, if the type $\prod_{x_1,x_2:X} [x_1] \equiv [x_2] \to \|x_1\, R\, x_2\|$ is inhabited.

Proposition 4.2. All quotients are weakly effective.

Proof. Let X be a type, R an equivalence relation on X and $x : X$. Consider the function $\|x\, R\, \_\| : X \to \mathcal{U}$, $\|x\, R\, \_\| = \lambda\, x'.\, \|x\, R\, x'\|$. We show that $\|x\, R\, \_\|$ is R-compatible. Let $x_1, x_2 : X$ with $x_1\, R\, x_2$. We have $x\, R\, x_1 \leftrightarrow x\, R\, x_2$ and therefore $\|x\, R\, x_1\| \leftrightarrow \|x\, R\, x_2\|$. Since propositional truncations are propositions, using proposition extensionality, we conclude $\|xRx_1\| \equiv \|xRx_2\|$. We have constructed a term $p_x : \mathsf{compat}\, \|x\, R\, \_\|$, and therefore a function $\mathsf{lift}\, \|x\, R\, \_\|\, p_x : X/R \to \mathcal{U}$ (large elimination is fundamental in order to apply lift, since $\|x\, R\, \_\| : X \to \mathcal{U}$ and $\mathcal{U} : \mathcal{U}_1$). Moreover, $\mathsf{lift}\, \|x\, R\, \_\|\, p_x\, [y] \equiv \|x\, R\, y\|$ by its computation rule.

Let $[x_1] \equiv [x_2]$ for some $x_1, x_2 : X$. We have

$$\|x_1\, R\, x_2\| \equiv \mathsf{lift}\, \|x_1 R\, \_\, \|\, p_{x_1}\, [x_2] \equiv \mathsf{lift}\, \|x_1\, R\, \_\, \|\, p_{x_1}\, [x_1] \equiv \|x_1\, R\, x_1\|$$

and $x_1\, R\, x_1$ holds, since R is reflexive. □

Notice that the constructor $[\_]$ is not a split epimorphism for all quotients. The existence of a choice of representative for each equivalence class is a non-constructive principle, since it implies the *law of excluded middle*, i.e., the inhabitedness of the following type:

$$\mathsf{LEM} = \prod_{\{X:\mathcal{U}\}} \mathsf{isProp}\, X \to X + \neg X,$$

where $\neg X = X \to 0$.

Proposition 4.3. Suppose that $[\_]$ is a split epimorphism for all quotients. Then LEM is inhabited.

Proof. Let X be a type together with a proof of $\mathsf{isProp}\, X$. We consider the equivalence relation R on Bool, $x_1\, R\, x_2 = x_1 \equiv x_2 + X$. By $[\_]$ being a split epimorphism, we obtain $\| \sum_{\mathsf{rep}:\mathsf{Bool}/R \to \mathsf{Bool}} \prod_{q:\mathsf{Bool}/R} [\mathsf{rep}\, q] \equiv q \|$. Using the elimination principle of propositional truncation, it is sufficient to construct a constant function of type:

$$\left(\sum_{\mathsf{rep}:\mathsf{Bool}/R \to \mathsf{Bool}} \prod_{q:\mathsf{Bool}/R} [\mathsf{rep}\, q] \equiv q \right) \to X + \neg X.$$

Let $\mathsf{rep} : \mathsf{Bool}/R \to \mathsf{Bool}$ with $[\mathsf{rep}\, q] \equiv q$ for all $q : \mathsf{Bool}/R$. We have $[\mathsf{rep}\,[x]] \equiv [x]$ for all $x : \mathsf{Bool}$, which by Proposition 4.2 implies $\|\mathsf{rep}\,[x]\; R\; x\|$.

Note now that the following implication (a particular instance of axiom of choice on Bool) holds:

$$\begin{aligned} &\mathsf{acBool} : \left(\prod_{x:\mathsf{Bool}} \|\mathsf{rep}\,[x]\; R\; x\|\right) \to \left\|\prod_{x:\mathsf{Bool}} \mathsf{rep}\,[x]\; R\; x\right\| \\ &\mathsf{acBool}\, r = \mathsf{lift}_2\,(\lambda r_1\, r_2.\, |\lambda x.\, \mathsf{if}\; x\; \mathsf{then}\; r_1\; \mathsf{else}\; r_2|)\; p\; (r\; \mathsf{true})\; (r\; \mathsf{false}), \end{aligned}$$

where $\mathsf{if\;true\;then}\; r_1\; \mathsf{else}\; r_2 = r_1$ and $\mathsf{if\;false\;then}\; r_1\; \mathsf{else}\; r_2 = r_2$, and lift_2 is the two-argument version of lift. The compatibility proof p is immediate, since the return type is a proposition.

We now construct a function of type $\|\prod_{x:\mathsf{Bool}} \mathsf{rep}\,[x]\; R\; x\| \to X + \neg X$. It is sufficient to define a function $\left(\prod_{x:\mathsf{Bool}} \mathsf{rep}\,[x]\; R\; x\right) \to X + \neg X$ (it will be constant, since the type $X + \neg X$ is a proposition, if X is a proposition), so we suppose $\mathsf{rep}\,[x]\; R\, x$ for all $x : \mathsf{Bool}$. We analyze the (decidable) equality $\mathsf{rep}\,[\mathsf{true}] \equiv \mathsf{rep}\,[\mathsf{false}]$ on Bool. If it holds, then we have $\mathsf{true}\; R\; \mathsf{false}$ and therefore an inhabitant of X. If it does not hold, we have an inhabitant of $\neg X$: indeed, suppose $x : X$, then $\mathsf{true}\; R\, \mathsf{false}$, so $[\mathsf{true}] \equiv [\mathsf{false}]$ and therefore $\mathsf{rep}\,[\mathsf{true}] \equiv \mathsf{rep}\,[\mathsf{false}]$, which contradicts the hypothesis. □

We already noted that not all quotients are effective. In fact, postulating effectiveness for all quotients implies LEM (Maietti 1999). But the quotient we are considering in this paper, namely $\mathsf{D}\, X/{\approx}$ for a type X, is indeed effective. Notice that, by Proposition 4.2, it suffices to prove that $\|c_1 \approx c_2\| \to c_1 \approx c_2$ for all $c_1, c_2 : \mathsf{D}\, X$.

Lemma 4.1. For all types X and $c_1, c_2 : \mathsf{D}\, X$, there exists a constant endofunction on $c_1 \approx c_2$. Therefore, the type $\|c_1 \approx c_2\| \to c_1 \approx c_2$ is inhabited.

Proof. Let X be a type and $c_1, c_2 : \mathsf{D}\, X$. We consider the following function.

$$\begin{aligned} &\mathsf{canon}{\approx} : c_1 \approx c_2 \to c_1 \approx c_2 \\ &\mathsf{canon}{\approx}\,({\downarrow_\approx}\,(\mathsf{now}_\downarrow\, p_1)\, p_2) = {\downarrow_\approx}\,(\mathsf{now}_\downarrow\, p_1)\, p_2 \\ &\mathsf{canon}{\approx}\,({\downarrow_\approx}\,(\mathsf{later}_\downarrow\, p_1)\,(\mathsf{now}_\downarrow\, p_2)) = {\downarrow_\approx}\,(\mathsf{later}_\downarrow\, p_1)\,(\mathsf{now}_\downarrow p_2) \\ &\mathsf{canon}{\approx}\,({\downarrow_\approx}\,(\mathsf{later}_\downarrow\, p_1)\,(\mathsf{later}_\downarrow\, p_2)) = \mathsf{later}_\approx\,(\mathsf{canon}{\approx}\,({\downarrow_\approx}\, p_1\, p_2)) \\ &\mathsf{canon}{\approx}\,(\mathsf{later}_\approx\, p) = \mathsf{later}_\approx\,(\mathsf{canon}{\approx}\, p). \end{aligned}$$

The function $\mathsf{canon}{\approx}$ canonizes a given weak bisimilarity proof by maximizing the number of applications of the constructor $\mathsf{later}_\approx$. This function is indeed constant, i.e., one can prove $\prod_{p_1, p_2 : c_1 \approx c_2} p_1 \approxeq p_2$ for all $c_1, c_2 : \mathsf{D}\, X$, where the relation $\approxeq$ is strong bisimilarity on proofs of $c_1 \approx c_2$, coinductively defined by the rules:

$$\frac{}{{\downarrow_\approx}\, p_1\, p_2 \approxeq {\downarrow_\approx}\, p_1\, p_2} \qquad \frac{p_1 \approxeq p_2}{\mathsf{later}_\approx\, p_1 \approxeq \mathsf{later}_\approx\, p_2}$$

Similarly to extensionality of delayed computations, we assume that strongly bisimilar weak bisimilarity proofs are equal, i.e., that we have an inhabitant for

$$\approx\mathsf{Ext} = \prod_{\{X:\mathcal{U}\}} \prod_{\{c_1,c_2:\mathsf{D}\,X\}} \prod_{p_1,p_2:c_1\approx c_2} p_1 \approxeq p_2 \to p_1 \equiv p_2. \qquad \square$$

For the quotient $\mathsf{D}\,X/{\approx'}$, no result similar to Lemma 4.1 is needed. In fact, since the relation $\approx'$ is propositional, we immediately obtain that $\mathsf{D}\,X/{\approx'}$ is effective.

5. Multiplication: What goes wrong?

Consider now the type functor $\overline{\mathsf{D}}$, defined by $\overline{\mathsf{D}}\,X = \mathsf{D}\,X/{\approx}$. Let us try to equip it with a monad structure. Let X be a type. As the unit $\overline{\eta} : X \to \mathsf{D}\,X/{\approx}$, we can take $[\_] \circ \mathsf{now}$. But when we try to construct a multiplication $\overline{\mu} : \mathsf{D}\,(\mathsf{D}\,X/{\approx})/{\approx} \to \mathsf{D}\,X/{\approx}$, we get stuck immediately. Indeed, the multiplication $\overline{\mu}$ must be of the form $\mathsf{lift}\,\overline{\mu}'\,p$ for some $\overline{\mu}' : \mathsf{D}\,(\mathsf{D}\,X/{\approx}) \to \mathsf{D}\,X/{\approx}$ with $p : \mathsf{compat}\,\overline{\mu}'$, but we cannot define such $\overline{\mu}'$ and p. The problem lies in the coinductive nature of the delay datatype. A function of type $\mathsf{D}\,(\mathsf{D}\,X/{\approx}) \to \mathsf{D}\,X/{\approx}$ should send a converging computation to its converging value and a non-terminating one to the equivalence class of non-termination. This discontinuity makes constructing such a function problematic. Moreover, one can show that a right inverse of $[\_] : \mathsf{D}\,X \to \mathsf{D}\,X/{\approx}$, i.e., a canonical choice of representative for each equivalence class in $\mathsf{D}\,X/{\approx}$, is not definable (Nuo 2015, Ch. 5.4.3). Therefore, we cannot even construct $\overline{\mu}'$ as a composition $[\_] \circ \overline{\mu}''$ with $\overline{\mu}'' : \mathsf{D}\,(\mathsf{D}\,X/{\approx}) \to \mathsf{D}\,X$, since we do not know how to define $\overline{\mu}''(\mathsf{now}\,q)$ for $q : \mathsf{D}\,X/{\approx}$.

A function $\overline{\mu}'$ would be constructible, if the type $\mathsf{D}\,(\mathsf{D}\,X/{\approx})$ were a quotient of $\mathsf{D}\,(\mathsf{D}\,X)$ by the equivalence relation $\mathsf{D}\,{\approx}$ (remember that $\mathsf{D}\,{\approx}$ is a synonym of $\sim_{\approx}$, the functorial lifting of $\approx$ from $\mathsf{D}\,X$ to $\mathsf{D}\,(\mathsf{D}\,X)$). In fact, the function $[\_] \circ \mu : \mathsf{D}\,(\mathsf{D}\,X) \to \mathsf{D}\,X/{\approx}$ is $\mathsf{D}\,{\approx}$-compatible, since $x_1(\mathsf{D}\,{\approx})x_2 \to \mu\,x_1 \approx \mu\,x_2$, and therefore the elimination principle would do the job. But how 'different' are $\mathsf{D}\,(\mathsf{D}\,X/{\approx})$ and the quotient $\mathsf{D}\,(\mathsf{D}\,X)/\mathsf{D}\,{\approx}$? More generally, how 'different' are $\mathsf{D}\,(X/R)$ and the quotient $\mathsf{D}\,X/\mathsf{D}\,R$, for a given type X and equivalence relation R on X?

5.1. *A limitation of quotients*

A function $\theta^{\mathsf{D}} : \mathsf{D}\,X/\mathsf{D}\,R \to \mathsf{D}\,(X/R)$ always exists, $\theta^{\mathsf{D}} = \mathsf{lift}\,(\mathsf{D}\,[\_])\,p$. The compatibility proof p follows directly from $c_1(\mathsf{D}\,R)c_2 \to \mathsf{D}\,[\_]\,c_1 \sim \mathsf{D}\,[\_]\,c_2$. But an inverse function $\psi^{\mathsf{D}} : \mathsf{D}\,(X/R) \to \mathsf{D}\,X/\mathsf{D}\,R$ is not definable. This phenomenon can be spotted more generally in non-wellfounded trees, i.e., the canonical function $\theta^{T} : T\,X/T\,R \to T\,(X/R)$ does not have an inverse, if $T\,X$ is coinductively defined, where $T\,R$ is the functorial lifting of R to $T\,X$. On the other hand, a large class of purely inductive types, namely, the datatypes of wellfounded trees where branching is finite, is free of this problem. As an example, for binary trees the inverse $\psi^{\mathsf{BTree}} : \mathsf{BTree}\,(X/R) \to \mathsf{BTree}\,X/\mathsf{BTree}\,R$ of

$\theta^{\mathsf{BTree}} : \mathsf{BTree}\,X/\mathsf{BTree}\,R \to \mathsf{BTree}\,(X/R)$ is defined as follows:

$$\begin{aligned}
&\psi^{\mathsf{BTree}} : \mathsf{BTree}\,(X/R) \to \mathsf{BTree}\,X/\mathsf{BTree}\,R\\
&\psi^{\mathsf{BTree}}\,(\mathsf{leaf}\,q) = \mathsf{lift}\,(\lambda\,x.\,[\mathsf{leaf}\,x])\,p_{\mathsf{leaf}}\,q\\
&\psi^{\mathsf{BTree}}\,(\mathsf{node}\,t_1\,t_2) = \mathsf{lift}_2\,(\lambda\,s_1\,s_2.\,[\mathsf{node}\,s_1\,s_2])\,p_{\mathsf{node}}\,(\psi^{\mathsf{BTree}}\,t_1)\,(\psi^{\mathsf{BTree}}\,t_2),
\end{aligned}$$

where lift_2 is the two-argument version of lift. The simple compatibility proofs p_{leaf} and p_{node} are omitted. Wellfounded non-finitely branching trees are affected by the same issues that non-wellfounded trees have. And in general, for a W-type (a general-form wellfounded tree type) T, the function $\theta^T : T\,X/T\,R \to T\,(X/R)$ cannot be inverted, since for function spaces the function $\theta^{Y\to} : (Y \to X)/(Y \to R) \to (Y \to X/R)$ cannot be inverted. Invertibility of the function $\theta^{Y\to} : (Y \to X)/(Y \to R) \to (Y \to X/R)$, for all types Y, X and equivalence relation R on X, has been analyzed in the Calculus of Inductive Constructions (Chicli et al. 2003). It turns out that surjectivity of $\theta^{Y\to}$ is logically equivalent to the full axiom of choice (AC)[†], i.e., the following type is inhabited:

$$\mathsf{AC} = \prod_{\{X,Y:\mathcal{U}\}}\ \prod_{P:X\to Y\to\mathcal{U}} \left(\prod_{x:X} \left\|\sum_{y:Y} P\,x\,y\right\|\right) \to \left\|\sum_{f:X\to Y}\ \prod_{x:X} P\,x\,(f\,x)\right\|.$$

Together with weak effectiveness (Proposition 4.2), AC implies not only surjectivity of $\theta^{Y\to}$, but also the existence of an inverse $\psi^{Y\to} : (Y \to X/R) \to (Y \to X)/(Y \to R)$. We refrain from proving these facts, but we prove Lemma 6.1 and Proposition 6.1, which are weaker statements, but have analogous proofs.

5.2. *A costly solution*

The existence of an inverse $\psi^{Y\to}$ of $\theta^{Y\to}$ would immediately allow us to define the bind operation for $\overline{\mathsf{D}}$. Let us consider the case where X is $\mathsf{D}\,X$ and R is weak bisimilarity, so $\psi^{Y\to} : (Y \to \mathsf{D}\,X/{\approx}) \to (Y \to \mathsf{D}\,X)/(Y \to {\approx})$. We define

$$\begin{aligned}
&\overline{\mathsf{bind}} : (Y \to \mathsf{D}\,X/{\approx}) \to \mathsf{D}\,Y/{\approx} \to \mathsf{D}\,X/{\approx}\\
&\overline{\mathsf{bind}}\,f\,q = \mathsf{lift}_2\,(\lambda\,g\,c.\,[\mathsf{bind}\,g\,c])\,p\,(\psi^{Y\to}\,f)\,q,
\end{aligned}$$

where bind is the bind operation of the unquotiented delay monad. The compatibility proof p is obtained from the fact that $\mathsf{bind}\,g_1\,c_1 \approx \mathsf{bind}\,g_2\,c_2$ if $c_1 \approx c_2$ and $g_1\,y \approx g_2\,y$ for all $y : Y$.

AC is a controversial semi-classical axiom, generally not accepted in constructive systems (Martin-Löf 2006). We reject it too, since in our system the axiom of choice implies the law of excluded middle.

[†] Notice that AC is fundamentally different from the *intuitionistic* axiom of choice:

$$\prod_{\{X,Y:\mathcal{U}\}}\ \prod_{P:X\to Y\to\mathcal{U}} \left(\prod_{x:X} \sum_{y:Y} P\,x\,y\right) \to \sum_{f:X\to Y}\ \prod_{x:X} P\,x\,(f\,x)$$

which is provable in type theory.

Proposition 5.1. AC implies LEM.

Proof. Assume AC. With a proof analogous to that of Lemma 6.1, we can prove that the function $\lambda f. [\_] \circ f : (X \to Y) \to (X \to Y/R)$ is surjective, for any types X, Y and equivalence relation R on Y. In particular, given a type X and an equivalence relation R on X, we have that the type $\prod_{g:X/R \to X/R} \left\| \sum_{f:X/R\to X} [\_] \circ f \equiv g \right\|$ is inhabited. Instantiating g with the identity function on X/R, we obtain $\left\| \sum_{f:X/R\to X} \prod_{q:X/R} [f\, q] \equiv q \right\|$, i.e., the constructor $[\_]$ is a split epimorphism for all quotients X/R. By Proposition 4.3, this implies LEM. □

In the following sections, we show that the weaker axiom of countable choice is already enough for constructing a multiplication for $\overline{\mathsf{D}}$. Countable choice does not imply excluded middle and constructive mathematicians like it more (Troelstra and Van Dalen 1988, Ch. 4). On the other hand, there exist models of type theory in which countable choice does not hold (Coquand et al. 2017).

6. Axiom of countable choice and streams of quotients

The axiom of countable choice ($\mathsf{AC}\omega$) is a specific instance of AC, where the binary predicate P has its first argument in $\mathbb{N}$:

$$\mathsf{AC}\omega = \prod_{\{X:\mathcal{U}\}} \prod_{P:\mathbb{N}\to X\to\mathcal{U}} \left(\prod_{n:\mathbb{N}} \left\| \sum_{x:X} P\, n\, x \right\| \right) \to \left\| \sum_{f:\mathbb{N}\to X} \prod_{n:\mathbb{N}} P\, n\, (f\, n) \right\|.$$

We also introduce a logically equivalent formulation of $\mathsf{AC}\omega$ that will be used in Proposition 6.1:

$$\mathsf{AC}\omega_2 = \prod_{P:\mathbb{N}\to\mathcal{U}} \left(\prod_{n:\mathbb{N}} \|P\, n\| \right) \to \left\| \prod_{n:\mathbb{N}} P\, n \right\|.$$

Let X be a type and R an equivalence relation on it. We show that $\mathsf{AC}\omega$ implies the surjectivity of the function $[\_]^{\mathbb{N}} : (\mathbb{N} \to X) \to (\mathbb{N} \to X/R)$, $[f]^{\mathbb{N}}\, n = [f\, n]$. This in turn implies the definability of a function $\psi^{\mathbb{N}} : (\mathbb{N} \to X/R) \to (\mathbb{N} \to X)/(\mathbb{N} \to R)$ inverting the canonical function $\theta^{\mathbb{N}} = \mathsf{lift}\, [\_]^{\mathbb{N}}\, \mathsf{sound}^{\mathbb{N}}$, where

$$\mathsf{sound}^{\mathbb{N}} : \mathsf{compat}\, [\_]^{\mathbb{N}}$$
$$\mathsf{sound}^{\mathbb{N}}\, r = \mathsf{funext}\, (\lambda n. \mathsf{sound}\, (r\, n))$$

using $\mathsf{funext} : \mathsf{FunExt}$.

Lemma 6.1. Assume $\mathsf{ac}\omega : \mathsf{AC}\omega$. Then $[\_]^{\mathbb{N}}$ is surjective.

Proof. Given any $g : \mathbb{N} \to X/R$, we construct a term $e_g : \left\| \sum_{f:\mathbb{N}\to X} [f]^{\mathbb{N}} \equiv g \right\|$. Since we are assuming the principle of function extensionality, it is sufficient to find a term $e'_g : \left\| \sum_{f:\mathbb{N}\to X} \prod_{n:\mathbb{N}} [f\, n] \equiv g\, n \right\|$. Define $P : \mathbb{N} \to X \to \mathcal{U}$ by $P\, n\, x = [x] \equiv g\, n$. We take $e'_g = \mathsf{ac}\omega\, P\, (\lambda n. [\_]\mathsf{surj}\, (g\, n))$, with $[\_]\mathsf{surj}$ introduced in Proposition 4.1. □

Proposition 6.1. Assume $\mathsf{AC}\omega$. Then $\theta^{\mathbb{N}} : (\mathbb{N} \to X)/(\mathbb{N} \to R) \to (\mathbb{N} \to X/R)$ is invertible.

Proof. We construct a term

$$r : \sum_{\psi^{\mathbb{N}} : (\mathbb{N} \to X/R) \to (\mathbb{N} \to X)/(\mathbb{N} \to R)} \prod_{g : \mathbb{N} \to X/R} \theta^{\mathbb{N}} \, (\psi^{\mathbb{N}} \, g) \equiv g.$$

Given any $g : \mathbb{N} \to X/R$, we define

$$h'_g : \left(\sum_{f : \mathbb{N} \to X} [f]^{\mathbb{N}} \equiv g \right) \to \sum_{q : (\mathbb{N} \to X)/(\mathbb{N} \to R)} \theta^{\mathbb{N}} \, q \equiv g$$
$$h'_g \, (f, p) = \Big([f], \mathsf{trans} \, (\mathsf{lift}_\beta \, [\_]^{\mathbb{N}} \, \mathsf{sound}^{\mathbb{N}} \, f) \, p \Big).$$

The function h'_g is constant. Let $f_1, f_2 : \mathbb{N} \to X$ with $p_1 : [f_1]^{\mathbb{N}} \equiv g$ and $p_2 : [f_2]^{\mathbb{N}} \equiv g$. By uniqueness of identity proofs, it is sufficient to show $[f_1] \equiv [f_2]$. By symmetry and transitivity, we get $[f_1]^{\mathbb{N}} \equiv [f_2]^{\mathbb{N}}$. We construct the following series of implications:

$$\begin{aligned}
[f_1]^{\mathbb{N}} \equiv [f_2]^{\mathbb{N}} &\to \prod_{n : \mathbb{N}} [f_1 \, n] \equiv [f_2 \, n] \\
&\to \prod_{n : \mathbb{N}} \|(f_1 \, n) \, R \, (f_2 \, n)\| && \text{(by weak effectiveness)} \\
&\to \left\| \prod_{n : \mathbb{N}} (f_1 \, n) \, R \, (f_2 \, n) \right\| && \text{(by } \mathsf{AC}\omega \text{ and } \mathsf{AC}\omega \to \mathsf{AC}\omega_2\text{)} \\
&= \|f_1 \, (\mathbb{N} \to R) \, f_2\| \\
&\to [f_1] \equiv [f_2].
\end{aligned}$$

The last implication is given by the elimination principle of propositional truncation applied to sound, which is a constant function by uniqueness of identity proofs. Therefore, h'_g is constant and we obtain a function

$$h_g : \left\| \sum_{f : \mathbb{N} \to X} [f]^{\mathbb{N}} \equiv g \right\| \to \sum_{q : (\mathbb{N} \to X)/(\mathbb{N} \to R)} \theta^{\mathbb{N}} \, q \equiv g.$$

We get $h_g \, e_g : \sum_{q : (\mathbb{N} \to X)/(\mathbb{N} \to R)} \theta^{\mathbb{N}} \, q \equiv g$, with e_g constructed in Lemma 6.1. We take $r = (\lambda g. \, \mathsf{fst} \, (h_g \, e_g), \lambda g. \, \mathsf{snd} \, (h_g \, e_g))$ and $\psi^{\mathbb{N}} = \mathsf{fst} \, r$.

We now prove that $\psi^{\mathbb{N}} \, (\theta^{\mathbb{N}} \, q) \equiv q$ for all $q : (\mathbb{N} \to X)/(\mathbb{N} \to R)$. It is sufficient to prove this equality for $q = [f]$ with $f : \mathbb{N} \to X$. By the computation rule of quotients, we have to show $\psi^{\mathbb{N}} \, [f]^{\mathbb{N}} \equiv [f]$. This is true, since

$$\psi^{\mathbb{N}} \, [f]^{\mathbb{N}} = \mathsf{fst} \, (h_{[f]^{\mathbb{N}}} \, e_{[f]^{\mathbb{N}}}) \equiv \mathsf{fst} \, (h_{[f]^{\mathbb{N}}} \, |f, \mathsf{refl}|) \equiv \mathsf{fst} \, (h'_{[f]^{\mathbb{N}}} (f, \mathsf{refl})) = [f]. \qquad \square$$

Corollary 6.1. Assume $\mathsf{AC}\omega$. The type $\mathbb{N} \to X/R$ is the carrier of a quotient of $\mathbb{N} \to X$ by the equivalence relation $\mathbb{N} \to R$. The constructor is $[\_]^{\mathbb{N}}$. We have the following dependent eliminator and computation rule: for every family of types $Y : (\mathbb{N} \to X/R) \to \mathcal{U}_k$ and for every function $h : \prod_{f : \mathbb{N} \to X} Y \, [f]^{\mathbb{N}}$ with proof $p : \mathsf{dcompat}^{\mathbb{N}} \, h$, there exists a function

$\mathsf{lift}^{\mathbb{N}}\, h\, p : \prod_{g:\mathbb{N}\to X/R} Y\, g$ with the property that $\mathsf{lift}^{\mathbb{N}}\, h\, p\, [f]^{\mathbb{N}} \equiv h\, f$ for all $f : \mathbb{N} \to X$, where

$$\mathsf{dcompat}^{\mathbb{N}}\, h = \prod_{\{f_1, f_2 : \mathbb{N}\to X\}} \prod_{r : f_1\, (\mathbb{N}\to R)\, f_2} \mathsf{subst}\, Y\, (\mathsf{sound}^{\mathbb{N}}\, r)\, (h\, f_1) \equiv h\, f_2.$$

7. Multiplication: A solution using $\mathsf{AC}\omega$

We can now build the desired monad structure on $\overline{\mathsf{D}}$ using the results proved in Section 6. In particular, we can define $\overline{\mu} : \mathsf{D}\,(\mathsf{D}\, X/{\approx})/{\approx} \to \mathsf{D}\, X/{\approx}$. We rely on $\mathsf{AC}\omega$.

7.1. *Delayed computations as streams*

In order to use the results of Section 6, we think of possibly non-terminating computations as streams. More precisely, let X be a type and $c : \mathsf{D}\, X$. Now c can be thought of as a stream $\varepsilon\, c : \mathbb{N} \to X + 1$ with at most one value element in the left summand X.

$$\begin{aligned}
&\varepsilon : \mathsf{D}\, X \to \mathbb{N} \to X + 1\\
&\varepsilon\, (\mathsf{now}\, x)\, \mathsf{zero} = \mathsf{inl}\, x\\
&\varepsilon\, (\mathsf{later}\, c)\, \mathsf{zero} = \mathsf{inr}\, \star\\
&\varepsilon\, (\mathsf{now}\, x)\, (\mathsf{suc}\, n) = \mathsf{inr}\, \star\\
&\varepsilon\, (\mathsf{later}\, c)\, (\mathsf{suc}\, n) = \varepsilon\, c\, n.
\end{aligned}$$

Conversely, from a stream $f : \mathbb{N} \to X + 1$, one can construct a computation $\pi\, f : \mathsf{D}\, X$. This computation corresponds to the ‘truncation’ of the stream to its first value in X.

$$\begin{aligned}
&\pi : (\mathbb{N} \to X + 1) \to \mathsf{D}\, X\\
&\pi\, f = \mathsf{case}\ f\ \mathsf{zero}\ \mathsf{of}\\
&\qquad \mathsf{inl}\, x \mapsto \mathsf{now}\, x\\
&\qquad \mathsf{inr}\, \star \mapsto \mathsf{later}\, (\pi\, (f \circ \mathsf{suc})).
\end{aligned}$$

We see that $\mathsf{D}\, X$ corresponds to a subset of $\mathbb{N} \to X + 1$ in the sense that, for all $c : \mathsf{D}\, X$, $\pi\, (\varepsilon\, c) \sim c$, and therefore $\pi(\varepsilon\, c) \equiv c$ by delayed computation extensionality.

Let R be an equivalence relation on X. It is the case that the canonical function $\theta^{+1} : (X + 1)/(R + 1) \to X/R + 1$ has an inverse ψ^{+1} whose construction is similar to the construction of ψ^{BTree} for binary trees in Section 5. Therefore, for all $q : \mathsf{D}\, (X/R)$, we have that $\pi\, (\theta^{+1} \circ (\psi^{+1} \circ \varepsilon\, q)) \equiv q$.

We define $[\_]^{\mathsf{D}} : \mathsf{D}\, X \to \mathsf{D}\, (X/R)$ by $[\_]^{\mathsf{D}} = \mathsf{D}\, [\_]$. This function is compatible with the relation $\mathsf{D}\, R$, i.e., there exists a term $\mathsf{sound}^{\mathsf{D}} : \mathsf{compat}\, [\_]^{\mathsf{D}}$.

Proposition 7.1. The type $\mathsf{D}\, (X/R)$ is the carrier of a quotient of $\mathsf{D}\, X$ by the equivalence relation $\mathsf{D}\, R$. The constructor is $[\_]^{\mathsf{D}}$ and we have the following dependent eliminator and computation rule: for every family of types $Y : \mathsf{D}\, (X/R) \to \mathcal{U}_k$ and for every function $h : \prod_{c:\mathsf{D}\, X} Y\, [c]^{\mathsf{D}}$ with $p : \mathsf{dcompat}^{\mathsf{D}}\, h$, there exists a function $\mathsf{lift}^{\mathsf{D}}\, h\, p : \prod_{q:\mathsf{D}\, (X/R)} Y\, q$ such that $\mathsf{lift}^{\mathsf{D}}\, h\, p\, [c]^{\mathsf{D}} \equiv h\, c$ for all $c : \mathsf{D}\, X$, where

$$\mathsf{dcompat}^{\mathsf{D}}\, h = \prod_{\{c_1,c_2 : \mathsf{D}\, X\}} \prod_{r : c_1 (\mathsf{D}\, R)\, c_2} \mathsf{subst}\, Y\, (\mathsf{sound}^{\mathsf{D}}\, r)\, (h\, c_1) \equiv h\, c_2.$$

Proof. We only define the dependent eliminator. Let $h : \prod_{x:\mathsf{D}\,X} Y\, [x]^{\mathsf{D}}$ with $p : \mathsf{dcompat}^{\mathsf{D}}\, h$, and $q : \mathsf{D}\,(X/R)$. Let $g : \mathbb{N} \to (X+1)/(R+1)$, $g = \psi^{+1} \circ \varepsilon\, q$, so $\pi\, (\theta^{+1} \circ g) \equiv q$.

We will prove $Y(\pi\, (\theta^{+1} \circ g))$. By Corollary 6.1, it is sufficient to construct a function $h' : \prod_{f:\mathbb{N}\to X+1} Y(\pi\, (\theta^{+1} \circ [f]^{\mathbb{N}}))$ together with a proof $r : \mathsf{dcompat}^{\mathbb{N}}\, h'$. One can easily construct a proof $s : [\pi\, f]^{\mathsf{D}} \equiv \pi\, (\theta^{+1} \circ [f]^{\mathbb{N}})$, so we take $h'\, f = \mathsf{subst}\, Y\, s\, (h\, (\pi\, f))$. A proof $r : \mathsf{dcompat}^{\mathbb{N}}\, h'$ can be constructed by observing that, for all $f_1, f_2 : \mathbb{N} \to X+1$ satisfying $f_1\, (\mathbb{N} \to R+1)\, f_2$, one can prove $\pi\, f_1\, (\mathsf{D}\, R)\, \pi\, f_2$. □

7.2. *Construction of* $\overline{\mu}$

Using the elimination rule of the quotient $\mathsf{D}\,(X/R)$ defined in Proposition 7.1, we can finally define the multiplication $\overline{\mu}$ of $\overline{\mathsf{D}}$.

$$\begin{tikzcd}
 & \mathsf{D}\,(\mathsf{D}\,X) \arrow[r, "\mu"] \arrow[dl, "{[\_]}"'] \arrow[d, "{[\_]^{\mathsf{D}} = \mathsf{D}\,[\_]}"] & \mathsf{D}\,X \arrow[d, "{[\_]}"] \\
\mathsf{D}\,(\mathsf{D}\,X)/\mathsf{D}\approx \arrow[r, bend left, "\theta^{\mathsf{D}}"] \arrow[r, phantom, "\cong"] & \mathsf{D}\,(\mathsf{D}\,X/\approx) \arrow[l, bend left, "\psi^{\mathsf{D}}"] \arrow[r, "{\mathsf{lift}^{\mathsf{D}}\,([\_]\circ\mu)\,p}"] \arrow[d, "{[\_]}"'] & \mathsf{D}\,X/\approx \\
 & \mathsf{D}\,(\mathsf{D}\,X/\approx)/\approx \arrow[ur, "{\overline{\mu} = \mathsf{lift}\,(\mathsf{lift}^{\mathsf{D}}\,([\_]\circ\mu)\,p)\,p'}"'] &
\end{tikzcd}$$

To make sense of the above diagram, we must construct construct two compatibility proofs $p : \mathsf{compat}^{\mathsf{D}}\,([\_]\circ\mu)$ and $p' : \mathsf{compat}\,(\mathsf{lift}^{\mathsf{D}}\,([\_]\circ\mu)\,p)$, where $\mathsf{compat}^{\mathsf{D}}$ is the non-dependent version of $\mathsf{dcompat}^{\mathsf{D}}$.

The first proof is easy, since $c_1(\mathsf{D}\approx)c_2 \to \mu\, c_1 \approx \mu\, c_2$ for all $c_1, c_2 : \mathsf{D}\,(\mathsf{D}\,X)$.

It is considerably more complicated to prove compatibility of the second function. Let $q_1, q_2 : \mathsf{D}\,(\mathsf{D}\,X/\approx)$. We have to show $q_1 \approx q_2 \to \mathsf{lift}^{\mathsf{D}}\,([\_]\circ\mu)\,p\,q_1 \equiv \mathsf{lift}^{\mathsf{D}}\,([\_]\circ\mu)\,p\,q_2$. By the elimination principle of the quotient $\mathsf{D}\,(\mathsf{D}\,X/\approx)$, described in Proposition 7.1, it is sufficient to prove $[x_1]^{\mathsf{D}} \approx [x_2]^{D} \to \mathsf{lift}^{\mathsf{D}}\,([\_]\circ\mu)\,p\,[c_1]^{D} \equiv \mathsf{lift}^{\mathsf{D}}\,([\_]\circ\mu)\,p\,[c_2]^{D}$ for some $c_1, c_2 : \mathsf{D}\,(\mathsf{D}\,X)$. Applying the computation rule of the quotient $\mathsf{D}\,(\mathsf{D}\,X/\approx)$ and spelling out the definition of the constructor $[\_]^{\mathsf{D}}$, it remains to show $\mathsf{D}\,[\_]\,c_1 \approx \mathsf{D}\,[\_]\,c_2 \to [\mu\, c_1] \equiv [\mu\, c_2]$, which holds, if one can prove $\mathsf{D}\,[\_]\,c_1 \approx \mathsf{D}\,[\_]\,c_2 \to \mu\, c_1 \approx \mu\, c_2$. This is provable thanks to Lemma 4.1. It is easy to see why Lemma 4.1 is important for completing the compatibility proof of $\mathsf{lift}^{\mathsf{D}}\,([\_]\circ\mu)\,p$. The difficult case in the proof of $\mathsf{D}\,[\_]\,c_1 \approx \mathsf{D}\,[\_]\,c_2 \to \mu\, c_1 \approx \mu\, c_2$ is the case, where $c_1 = \mathsf{now}\, y_1$ and $c_2 = \mathsf{now}\, y_2$, so we are given an assumption of type $[y_1] \equiv [y_2]$. From this, by Lemma 4.1, we obtain $\mu\,(\mathsf{now}\, y_1) = y_1 \approx y_2 = \mu\,(\mathsf{now}\, y_2)$.

Theorem 7.1. Assuming $\mathsf{AC}\omega$, the type functor $\overline{\mathsf{D}}$, defined by $\overline{\mathsf{D}}\,X = \mathsf{D}\,X/{\approx}$, is a monad.

8. A monad or an arrow?

Hughes (2000) has proposed arrows as a generalization of monads. Jacobs et al. (2009) have sorted out their mathematical theory.

We have seen that it takes a semi-classical principle to show that quotienting the functor D by weak bisimilarity preserves its monad structure. In contrast, quotienting the corresponding profunctor KD, defined by $\mathsf{KD}\,X\,Y = X \to \mathsf{D}\,Y$, by pointwise weak bisimilarity can easily be shown to preserve its (strong) arrow structure (whose Freyd category is isomorphic to the Kleisli category of the monad) without invoking such principles.

Indeed, the arrow structure on KD is given by $\mathsf{pure} : (X \to Y) \to \mathsf{KD}\,X\,Y$, $\mathsf{pure}\,f = \eta \circ f$ and $\lll : \mathsf{KD}\,Y\,Z \to \mathsf{KD}\,X\,Y \to \mathsf{KD}\,X\,Z$, $\ell \lll k = \mathsf{bind}\,\ell \circ k$.

Now, define the quotiented profunctor by $\overline{\mathsf{KD}}\,X\,Y = (X \to \mathsf{D}\,Y)/(X \to {\approx})$. We can define $\overline{\mathsf{pure}} : (X \to Y) \to \overline{\mathsf{KD}}\,X\,Y$ straightforwardly by $\overline{\mathsf{pure}}\,f = [\mathsf{pure}\,f]$. But we can also construct $\overline{\lll} : \overline{\mathsf{KD}}\,Y\,Z \to \overline{\mathsf{KD}}\,X\,Y \to \overline{\mathsf{KD}}\,X\,Z$ as $\ell \mathbin{\overline{\lll}} k = \mathsf{lift}_2\,(\lll)\,p\,\ell\,k$, where p is an easy proof of $\ell_1\,(Y \to {\approx})\,\ell_2 \to k_1\,(X \to {\approx})\,k_2 \to (\ell_1 \lll k_1)\,(X \to {\approx})\,(\ell_2 \lll k_2)$.

This works entirely painlessly, as there is no need in this construction for a coercion $(X \to Y/{\approx}) \to (X \to Y)/(X \to {\approx})$ (cf. the discussion above in Section 5). From the beginning, we quotient the relevant function types here rather than their codomains.

There are some further indications that quotienting the arrow may be a sensible alternative to quotienting the monad. In particular, the work by Cockett et al. (2012) suggests that working with finer quotients of the arrow considered here may yield a setting for dealing with computational complexity rather than computability constructively.

9. Quotiented delay delivers free ωcppos

In this section, we show that the type $\mathsf{D}\,X/{\approx}$ is the free ω-complete pointed partial order over X. First we review some definitions.

9.1. *Preliminaries*

A *partially ordered set*, or *poset*, is a type X with a binary relation $\leqslant : X \to X \to \mathcal{U}$ which is reflexive, transitive and antisymmetric. We also require the binary relation to be propositional, i.e., we ask for the type $\mathsf{isProp}\,(x \leqslant y)$ to be inhabited, for all $x, y : X$. Notice that this requirement agrees with the categorical view of posets as categories with at most one arrow between any two objects. A poset $(X, \leqslant)$ is *pointed* if it has a least element, i.e., if there exists $\bot : X$ with $\bot \leqslant x$ for all $x : X$.

A morphism between two posets $(X, \leqslant)$ and $(Y, \sqsubseteq)$ is an *order-preserving* function $f : X \to Y$, that is $f\,x_1 \sqsubseteq f\,x_2$ if $x_1 \leqslant x_2$. A morphism between two pointed posets $(X, \leqslant, \bot_X)$ and $(Y, \sqsubseteq, \bot_Y)$ is *strict*, if it preserves the least element, i.e., $f\,\bot_X \sqsubseteq \bot_Y$.

A *ω-complete partial order* is a poset $(X, \leqslant)$ in which every chain, i.e., an increasing stream, has a supremum. This means that, given a stream $s : \mathbb{N} \to X$ with $s\,n \leqslant s\,(\mathsf{suc}\,n)$ for all $n : \mathbb{N}$, there exists an element $\bigcup s : X$, which is an upper bound for s, i.e., $s\,n \leqslant \bigcup s$ for all $n : \mathbb{N}$, and $\bigcup s \leqslant x$ for all $x : X$ that are upper bounds for s. We define $\mathsf{isIncr}\,s = \prod_{n:\mathbb{N}} s\,n \leqslant s\,(\mathsf{suc}\,n)$. A ω-complete pointed partial order is a partial order that is both pointed and ω-complete. From now on we refer to a ω-complete pointed partial orders simply as ωcppos.

A morphism between two ωcppos $(X, \leqslant, \bot_X, \bigcup)$ and $(Y, \sqsubseteq, \bot_Y, \bigsqcup)$ is an order-preserving and strict function $f : X \to Y$ that also preserves joins of chains, i.e., $f\,(\bigcup s) \sqsubseteq \bigsqcup(f \circ s)$. A ωcppo morphism is also called a *continuous* function.

The *free ωcppo over a type X* is a ωcppo $(\widehat{X}, \leqslant, \bot_{\widehat{X}}, \bigcup)$ such that there exists an injection function $i : X \to \widehat{X}$ and the following universal property holds. Given a ωcppo $(Y, \sqsubseteq, \bot_Y, \bigsqcup)$ and a function $f : X \to Y$, there exists a unique continuous function $\widehat{f} : (\widehat{X}, \leqslant, \bot_{\widehat{X}}, \bigcup) \to (Y, \sqsubseteq, \bot_Y, \bigsqcup)$ such that $\widehat{f}\,(i\,x) \equiv f\,x$ for all $x : X$.

Classically, the free ωcppo over X is the maybe datatype $X + 1$, which is typically pictured as a flat domain as follows:

$$\begin{tikzcd}[arrows=dash]
\mathsf{inl}\,x_0 & \mathsf{inl}\,x_1 & \mathsf{inl}\,x_2 & \cdots \\
 & & \mathsf{inr}\,* \arrow[ull, "\leqslant"] \arrow[ul, "\leqslant"] \arrow[u, "\leqslant"] \arrow[ur, "\leqslant"] &
\end{tikzcd}$$

where $x_0, x_1, \ldots$ are the inhabitants of X. Notice that constructively $X + 1$ is not a ωcppo. In fact, there is no way of constructing joins of general chains, since this is equivalent to a variant of the limited principle of omniscience (LPO): given a chain $s : \mathbb{N} \to X + 1$, either $s\,n \equiv \mathsf{inr}\,*$ for all $n : \mathbb{N}$, or there exists $n : \mathbb{N}$ and $x : X$ such that $s\,n \equiv \mathsf{inl}\,x$.

9.2. *Free ωcppo structure up to $\approx$*

In this subsection, we show that the type $\mathsf{D}\,X$ is endowed with a ωcppo structure up to $\approx$. Moreover, it is the free ωcppo up to $\approx$ over X. The construction performed in this subsection will be lifted to the quotient $\mathsf{D}\,X/{\approx}$ in Section 9.3. Following Capretta (2005), we introduce an information order 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 when $c_1 \approx c_2$, but also when c_1 has some (possibly infinitely many) laters more than c_2. The relation $\sqsubseteq$ is reflexive and transitive. Moreover, it is antisymmetric up to $\approx$, i.e., $c_1 \sqsubseteq c_2 \to c_2 \sqsubseteq c_1 \to c_1 \approx c_2$, for all $c_1, c_2 : \mathsf{D}\,X$. Notice also that the relation $\sqsubseteq$ is not propositional. The least element is the non-terminating computation never, corecursively defined as $\mathsf{never} = \mathsf{later}\,\mathsf{never}$.

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 $\omega\mathsf{race}$. The latter constructs the first converging element of a stream 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 stream $s : \mathbb{N} \to \mathsf{D}\,X$, a number $n : \mathbb{N}$ and a computation $c : \mathsf{D}\,X$, constructs the first converging element of the stream $s' : \mathbb{N} \to \mathsf{D}\,X$, 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 the stream s is increasing, then $\omega\mathsf{race}\,s$ is the join of s, i.e., the following terms exist:

$$\begin{aligned}
&\omega\mathsf{raceisUB} : \prod_{s:\mathbb{N}\to\mathsf{D}\,X}\ \prod_{i:\mathsf{isIncr}\,s}\ \prod_{n:\mathbb{N}} s\,n \sqsubseteq \omega\mathsf{race}\,s\\
&\omega\mathsf{raceisSupremum} : \prod_{s:\mathbb{N}\to\mathsf{D}\,X}\ \prod_{i:\mathsf{isIncr}\,s}\ \prod_{c:\mathsf{D}\,X} \left(\prod_{n:\mathbb{N}} s\,n \sqsubseteq c\right) \to \omega\mathsf{race}\,s \sqsubseteq c.
\end{aligned}$$

So far, we have showed that $(\mathsf{D}\,X, \sqsubseteq, \mathsf{never}, \omega\mathsf{race})$ is a ωcppo up to $\approx$. We prove that it is the free one over X. Let $(Y, \leqslant, \bot, \cup)$ be a ωcppo and $f : X \to Y$ a function. Every computation in $\mathsf{D}\,X$ defines a stream in Y.

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

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

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

Since the latter is increasing wrt. $\leqslant$, it is possible to extend the function f to a function $\widehat{f} : \mathsf{D}\,X \to Y$, $\widehat{f}\,c = \bigcup(\overline{f}\,c)$. We have that $\widehat{f}\,(\mathsf{now}\,x) \equiv f\,x$. Moreover $\widehat{f}$ is continuous, i.e., the following terms exist:

$$\begin{aligned} \mathsf{hatOrderpreserving} &: \prod_{c_1,c_2:\mathsf{D}\,X} c_1 \sqsubseteq c_2 \to \widehat{f}\,c_1 \leqslant \widehat{f}\,c_2, \\ \mathsf{hatStrict} &: \widehat{f}\,\mathsf{never} \leqslant \bot, \\ \mathsf{hatContinuous} &: \prod_{s:\mathbb{N}\to\mathsf{D}\,X}\ \prod_{i:\mathsf{isIncr}\,s} \widehat{f}\,(\omega\mathsf{race}\,s) \leqslant \bigcup(\widehat{f}\circ s). \end{aligned} \tag{1}$$

The last statement makes sense because, if s is increasing, then $\widehat{f}\circ s$ is also increasing, thanks to $\mathsf{hatOrderpreserving}$. Moreover, $\widehat{f}$ is $\approx$-compatible and it is the unique $\approx$-compatible map of the form $g : \mathsf{D}\,X \to Y$ that satisfies the inequalities in (1) and such that $g\,(\mathsf{now}\,x) \equiv f\,x$.

9.3. *Lifting the construction to* $\mathsf{D}\,X/\!\approx$

The first step we need to perform in order to lift all the constructions of Section 9.2 to $\mathsf{D}\,X/\!\approx$ is the lifting of the relation $\sqsubseteq$. Unfortunately, this cannot be done directly. In fact, if we try to define a binary relation $\sqsubseteq_\approx$ on $\mathsf{D}\,X/\!\approx$ as follows:

$$\begin{aligned} &\sqsubseteq_\approx : \mathsf{D}\,X/\!\approx\, \to \mathsf{D}\,X/\!\approx\, \to \mathcal{U} \\ &\sqsubseteq_\approx\, = \mathsf{lift}_2 \sqsubseteq p, \end{aligned}$$

we realize that we need to construct a term p inhabiting the type $\prod_{\{c_1,c_2,d_1,d_2:\mathsf{D}\,X\}} c_1 \approx d_1 \to c_2 \approx d_2 \to c_1 \sqsubseteq c_2 \equiv d_1 \sqsubseteq d_2$, which is not a true statement. In fact, let $c_1 = c_2 = \mathsf{now}\,x$ and $d_1 = d_2 = \mathsf{later}\,(\mathsf{now}\,x)$, then the type $c_1 \sqsubseteq c_2$ is a proposition, while the type $d_1 \sqsubseteq d_2$ is not.

In order to overcome this issue, we lift the propositional truncation of the relation $\sqsubseteq$ instead of the relation $\sqsubseteq$ directly, as follows:

$$\begin{aligned} &\sqsubseteq_\approx : \mathsf{D}\,X/\!\approx\, \to \mathsf{D}\,X/\!\approx\, \to \mathcal{U} \\ &\sqsubseteq_\approx\, = \mathsf{lift}_2\,(\lambda\,c_1,c_2.\,\|c_1 \sqsubseteq c_2\|)\,p, \end{aligned}$$

where p is a proof of $\prod_{\{c_1,c_2,d_1,d_2:\mathsf{D}\,X\}} c_1 \approx d_1 \to c_2 \approx d_2 \to \|c_1 \sqsubseteq c_2\| \equiv \|d_1 \sqsubseteq d_2\|$, which can be proved with the help of proposition extensionality. The relation $\sqsubseteq_\approx$ is propositional and the proofs of reflexivity, transitivity and antisymmetry up to $\approx$ of the relation $\sqsubseteq$ lift straightforwardly to the relation $\sqsubseteq_\approx$.

Remark 9.1. There is an alternative way of lifting $\sqsubseteq$ to $\mathsf{D}\,X/\!\approx$. Following Benton et al. (2009), we introduce a binary relation $\sqsubseteq'$ on $\mathsf{D}\,X$:

$$\frac{c \downarrow x}{\mathsf{now}\,x \sqsubseteq' c} \qquad \frac{c_1 \sqsubseteq' c_2}{\mathsf{later}\,c_1 \sqsubseteq' \mathsf{later}\,c_2} \qquad \frac{c \sqsubseteq' \mathsf{now}\,x}{\mathsf{later}\,c \sqsubseteq' \mathsf{now}\,x}$$

Notice the similarity with the definition of $\approx'$ in Remark 3.1. The relation $\sqsubseteq'$ is equivalent to $\sqsubseteq$, but it is propositional. This implies that $\sqsubseteq'$ is liftable to $\mathsf{D}\,X/\approx$:

$$\begin{aligned}\sqsubseteq'_{\approx} &: \mathsf{D}\,X/\approx\ \to \mathsf{D}\,X/\approx\ \to \mathcal{U}\\ \sqsubseteq'_{\approx} &= \mathsf{lift}_2 \sqsubseteq' p,\end{aligned}$$

where p is a proof of $\prod_{\{c_1,c_2,d_1,d_2:\mathsf{D}\,X\}} c_1 \approx d_1 \to c_2 \approx d_2 \to c_1 \sqsubseteq' c_2 \equiv d_1 \sqsubseteq' d_2$, which is a true statement. We prefer to work with $\sqsubseteq$ instead of $\sqsubseteq'$ for the same reasons specified in Remark 3.1 about $\approx$ and $\approx'$.

We now lift the operator $\omega\mathsf{race}$ to the quotient. We find ourselves in a situation similar to the one described in Section 5, where we noticed that infinite datatypes (in this case the type of streams) do not commute with quotienting. To deal with this issue we rely on the axiom of countable choice, more precisely on the eliminator $\mathsf{lift}^{\mathbb{N}}$ described in Corollary 6.1.

$$\begin{aligned}\omega\mathsf{race}_{\approx} &: \prod_{s:\mathbb{N}\to\mathsf{D}\,X/\approx} \mathsf{isIncr}_{\approx}\, s \to \mathsf{D}\,X/\approx\\ \omega\mathsf{race}_{\approx} &= \mathsf{lift}^{\mathbb{N}}\,(\lambda s.\,\lambda i.\,[\omega\mathsf{race}\, s])\, p,\end{aligned}$$

where p is a proof of compatibility of the function $\lambda s.\,\lambda i.\,[\omega\mathsf{race}\, s]$ with the relation $\mathbb{N} \to \approx$, which is indeed the case since the input stream is increasing wrt. the relation $\sqsubseteq_{\approx}$. The proofs attesting that $(\mathsf{D}\,X/\approx, \sqsubseteq_{\approx}, [\mathsf{never}], \omega\mathsf{race}_{\approx})$ is the free ωcppo over X are obtained by directly lifting the corresponding proofs described in Section 9.2 to $\mathsf{D}\,X/\approx$ with the help of the elimination principle constructed in Corollary 6.1. For the technical details, we refer to our full Agda formalization.

Theorem 9.1. Assuming $\mathsf{AC}\omega$, the type $\mathsf{D}\,X/\approx$ is the free ωcppo on X.

10. Partiality in homotopy type theory

The quotiented delay monad constitutes a possible way of representing partiality as an effect in type theory. Recently, Altenkirch et al. (2017) have constructed another datatype A for partiality in homotopy type theory. Their construction makes use of higher inductive–inductive types and resembles the implementation of Cauchy reals in the HoTT book (Univalent Foundations Program 2013, Ch. 11.3). The datatype A delivers free ωcppos by construction and it carries a monad structure without recourse to choice principles. Higher inductive–inductive types, rather than ordinary higher inductive types, are needed because the join constructor $\cup$ takes as argument a proof that a given stream is increasing. So, the type $\mathsf{A}\,X$ has to be introduced mutually with the partial order $\leqslant$ on it. Altenkirch et al. proved that $\mathsf{A}\,X$ is isomorphic to $\mathsf{D}\,X/\approx$ under the assumption of countable choice.

In this section, we present yet another datatype for partiality in homotopy type theory, which does not make use of choice principles or higher inductive–inductive definitions. It is constructed using ordinary higher inductive types (Univalent Foundations Program 2013, Ch. 6.13). As a consequence, our partiality datatype can be directly implemented in proof assistants such as Coq, which currently lack support for inductive–inductive types, and may be added to the HoTT library (Bauer et al. 2017). The datatype that we present in this section is isomorphic to A, and therefore, under the assumption of countable choice, also isomorphic to the quotiented delay datatype.

Our construction is based on the implementation of free countably complete join semilattices as higher inductive types. A *countably complete join semilattice* is a partially ordered set $(X, \leqslant)$ with a bottom element $\bot : X$ and a countable join operation $\bigvee : (\mathbb{N} \to X) \to X$. Notice that the join operation $\bigvee$ is defined for all streams, not just the increasing ones. A *countably complete join semilattice morphism* between countably complete join semilattices X and Y is a monotone function between X and Y which preserves bottom and joins.

Countably complete join semilattices admit an equational presentation as an infinitary algebraic theory. In homotopy type theory, it is possible to introduce the free object of an algebraic theory as a higher inductive type. This procedure is exemplified in the construction of the free group over a type (Univalent Foundations Program 2013, Ch. 6.11). Let X be a type, the free countably complete join semilattice on X is defined similarly as the following higher inductive type:

$$\frac{x : X}{\eta\, x : \mathcal{P}_\infty X} \qquad \frac{}{\bot : \mathcal{P}_\infty X} \qquad \frac{s : \mathbb{N} \to \mathcal{P}_\infty X}{\bigvee s : \mathcal{P}_\infty X}$$

$$\frac{}{x \vee y \equiv y \vee x} \qquad \frac{}{x \vee (y \vee z) \equiv (x \vee y) \vee z} \qquad \frac{}{x \vee x \equiv x} \qquad \frac{}{x \vee \bot \equiv x}$$

$$\frac{}{\prod_{n:\mathbb{N}} s\, n \vee \bigvee s \equiv \bigvee s} \qquad \frac{}{\bigvee s \vee x \equiv \bigvee(\lambda n.\, s\, n \vee x)} \qquad \text{the 0-truncation constructor}$$

where the binary join operation is derived as $x \vee y = \bigvee(x, y, y, y, \ldots)$. We define $x \leqslant y = x \vee y \equiv y$.

The type $\mathcal{P}_\infty X$ is the free countably complete join semilattice on X by construction. In the types of its constructors, it is possible to identify the algebraic theory of countably complete join semilattices. The 0-truncation constructor, stating that the type $x \equiv y$ is a proposition for all $x, y : \mathcal{P}_\infty X$, forces $\mathcal{P}_\infty X$ to be a set, i.e., to satisfy the principle of uniqueness of identity proofs UIP. The dependent eliminator of $\mathcal{P}_\infty X$ is an induction principle from which freeness (the unique mapping property) can be derived.

It is a well-known fact that the free countably complete join semilattice on a type X is the countable powerset of X, i.e., the type whose elements are the subsets of X with countable cardinality. The order $\leqslant$ is the inclusion order.

We define $\mathsf{S} = \mathcal{P}_\infty 1$. This type has $\top = \eta\, *$ as its top element, as we can prove by induction that $x \leqslant \top$ for all $x : \mathsf{S}$. The type S is the countable powerset of 1. It is important to realize that S is not isomorphic to Bool. $\bot$ corresponds to the empty subset and $\top$ corresponds to the full set. We can prove that $x \not\equiv \top$ implies $x \equiv \bot$ for all $x : \mathsf{S}$. But we cannot decide whether $x \equiv \bot$ or $x \equiv \top$. For a general $s : \mathbb{N} \to \mathsf{S}$, even if $s\, n$ is either $\bot$ or $\top$ for all $n : \mathbb{N}$, we cannot decide whether $s\, n \equiv \top$ for at least one $n : \mathbb{N}$, unless we assume LPO.

S happens to be also the initial σ-frame, i.e., a countably complete join semilattice with finite meets which distribute over joins. In fact, $\top$ is the top element and binary meets can be defined by induction.

S has an interesting relation with the free ωcppo on 1. If the latter exists, then they are isomorphic.

Proposition 10.1. The free ωcppo on 1 is also the free countably complete join semilattice on 1.

Proof. Let $(X, \leqslant, \bot, \cup)$ be the free ωcppo on 1. We only need to construct a countable join operation $\bigvee$.

For any $x : X$, by the universal property of X, there exists a unique continuous map $f_x : X \to \sum_{y:X} x \leqslant y$, since the latter is a ωcppo over 1. We can define a binary join operation as $x \vee y = \mathsf{fst}\,(f_x\, y)$. The countable join operation is defined as $\bigvee s = \cup s'$, where the stream s' is the majorization of s defined inductively by $s'\,\mathsf{zero} = s\,\mathsf{zero}$ and $s'\,(\mathsf{suc}\, n) = s'\,n \vee s\,(\mathsf{suc}\, n)$. The stream s' is increasing and can be supplied as an argument of the join operation $\cup$ of X.

It is not difficult to show that the type X, together with the data described above, is a countably complete join semilattice on 1.

Let Y be another countably complete join semilattice on 1. Notice that Y is also a ωcppo on 1. Therefore, by the universal property of X, there exists a unique ωcppo morphism between X and Y. It is not difficult to prove that the latter is also a countably complete join semilattice morphism and, moreover, the only existing one. □

Notice that the free ωcppo on a general X is not the free countably complete join semilattice on X since it does not have binary joins. We noticed this already in Section 9.2 when we introduced the binary operation race on $\mathsf{D}\,X$. As a consequence, the majorization of a stream presented in the proof of Proposition 10.1 is not definable.

From Theorem 9.1 and Proposition 10.1, we have that S is isomorphic to $\mathsf{D}\,1/{\approx}$, under the assumption of countable choice. Moreover it is isomorphic to $\mathsf{A}\,1$.

We define $\mathsf{P_S} X = \sum_{x:\mathsf{S}} (x \equiv \top \to X)$. We show that $\mathsf{P_S}$ carries a monad structure without the requirement of choice principles.

Proposition 10.2. $\mathsf{P_S}$ is a monad.

Proof. Notice that $\mathsf{P_S}$ is a functor specified by a container (Abbott et al. 2005): the set of shapes is $S = \mathsf{S}$, while the set of positions is $P\,x = x \equiv \top$, for all $x : \mathsf{S}$. Therefore, $\mathsf{P_S}$ carries a monad structure if and only if it comes with certain extra structure (Ahman et al. 2014), namely

$$\begin{aligned}
\mathsf{e} &: \mathsf{S}, \\
\bullet &: \prod_{x:\mathsf{S}} (x \equiv \top \to \mathsf{S}) \to \mathsf{S}, \\
q_0 &: \prod_{x:\mathsf{S}} \prod_{v:x\equiv\top\to\mathsf{S}} x \bullet v \equiv \top \to x \equiv \top, \\
q_1 &: \prod_{x:\mathsf{S}} \prod_{v:x\equiv\top\to\mathsf{S}} \prod_{p:x\bullet v\equiv\top} v\,(q_0\, x\, v\, p) \equiv \top,
\end{aligned}$$

satisfying the equations

$$x \bullet (\lambda\_.\,\mathsf{e}) \equiv x, \qquad \mathsf{e} \bullet (\lambda\_.\, x) \equiv x,$$

$$(x \bullet v) \bullet (\lambda p.\, w\,(q_0\, x\, v\, p)\,(q_1\, x\, v\, p)) \equiv x \bullet (\lambda p.\, v\, p \bullet w\, p).$$

Notice that, in general, more equalities between positions are required to hold. In our case, these equations are all trivial, since the type $x \equiv \top$ is a proposition, for all $x : \mathsf{S}$.

We take $\mathsf{e} = \top$, while the function $\bullet$ is defined by induction on its first argument:

$$\begin{aligned} \top \bullet v &= v\,\mathsf{refl} \\ \bot \bullet v &= \bot \\ \bigvee s \bullet v &= \bigvee(\lambda n.\, s\,n \bullet v'\,n), \end{aligned}$$

where, in the last row, $v' : \prod_{n:\mathbb{N}} (s\,n \equiv \top \to \mathsf{S})$ is obtained from $v : \bigvee s \equiv \top \to \mathsf{S}$ by noticing that $s\,n \equiv \top$ implies $\bigvee s \equiv \top$, for all $n : \mathbb{N}$. It is not difficult to see that the the function $\bullet$ 'respects equality,' i.e., that terms made equal by the 1-constructors of S have the same image under $\bullet$.

The terms q_0 and q_1 are constructed by induction on their first argument $x : \mathsf{S}$. The equation $\mathsf{e} \bullet (\lambda\_.\, x) \equiv x$ holds definitionally. The other two equations are proved by induction on the argument $x : \mathsf{S}$. □

We can prove that, similarly to the quotiented delay monad, the monad P_S delivers free ωcppos. Instead of countable choice, we have to assume that S is the free ωcppo on 1. We know that for this assumption to hold, it suffices that the free ωcppo on 1 exists, by Proposition 10.1.

Proposition 10.3. If S is the free ωcppo on 1, then $\mathsf{P}_\mathsf{S} X$ is the free ωcppo on X.

Proof. We construct an ωcppo structure on the type $\mathsf{P}_\mathsf{S} X$:

—A partial-order relation on $\mathsf{P}_\mathsf{S} X$ is constructed as follows:

$$(x_1, f_1) \leqslant' (x_2, f_2) = \sum_{p : x_1 \leqslant x_2} \prod_{q : x_1 \equiv \top} f_1\, q \equiv f_2\, (\mathsf{le2equiveta}\, p\, q),$$

where $\mathsf{le2equiveta} : x_1 \leqslant x_2 \to x_1 \equiv \top \to x_2 \equiv \top$ is an easy consequence of $\top$ being the maximal element of the relation $\leqslant$.

—The bottom element is $(\bot, f)$, where $f : \bot \equiv \top \to X$ is the empty function, since the type $\bot \equiv \top$ is empty.

—Let t be a stream increasing wrt. $\leqslant'$. The function t is of the form $\langle s, f \rangle$, for some stream $s : \mathbb{N} \to \mathsf{S}$ increasing wrt. $\leqslant$ and some function $f : \prod_{n:\mathbb{N}} s\,n \equiv \top \to X$. The least upper bound of t is computed as $(\bigvee s, f')$, where $f' : \bigvee s \equiv \top \to X$ is constructed as follows. First one proves that from a proof of $\bigvee s \equiv \top$ one gets a proof of $\|\sum_{n:\mathbb{N}} s\,n \equiv \top\|$. A function from the latter type to X is given using the elimination principle of propositional truncation applied to the term f. This operation can be performed because the function f is constant, i.e., $f\,n\,p \equiv f\,m\,q$ for $n, m : \mathbb{N}$, $p : s\,n \equiv \top$ and $q : s\,m \equiv \top$, and that is the case because the stream s is increasing.

Moreover, there exists a function $h : X \to \mathsf{P}_\mathsf{S} X$, given by $h\,x = (\top, \lambda\_.\, x)$. It is not difficult to check that the type $\mathsf{P}_\mathsf{S} X$, together with the previous data, is a ωcppo on X.

Next, we are given an arbitrary ωcppo on X, let us call it Y. We have to construct a ωcppo morphism between $\mathsf{P}_\mathsf{S} X$ and Y. We give a sketch of this construction. The desired map is defined in two steps. First, we give a proof $p : \prod_{x:\mathsf{S}} (x \equiv \top \to X) \to Y$. Remember that, by hypothesis, the type S is the free ωcppo on 1 and therefore it has an associated induction principle derivable from the freeness property. The term p is constructed using this induction principle applied to $x : \mathsf{S}$.[‡] Second, we show that the uncurried version of p is continuous. Moreover, it is the only such map between $\mathsf{P_S}X$ and Y. $\square$

In the presence of higher inductive–inductive types, $\mathsf{A}\,1$ is the free ωcppo on 1. Therefore, the type $\mathsf{P_S}X$ is isomorphic to $\mathsf{A}\,X$. As a consequence, assuming countable choice, $\mathsf{P_S}X$ is isomorphic to $\mathsf{D}\,X/\approx$. We do not show it here, but one can construct the isomorphism between $\mathsf{P_S}X$ and $\mathsf{D}\,X/\approx$ also directly, without going through $\mathsf{A}\,X$, but still assuming countable choice.

By Proposition 10.2, we know that $\mathsf{P_S}$ is a monad. One can prove a stronger result: $\mathsf{P_S}$ is a partial map classifier in the sense of Mulry (1994), classifying specifically partial functions with a semidecidable domain of definedness. This means that maps in the Kleisli category of $\mathsf{P_S}$ are in one-to-one correspondence with maps in a category of partial maps. In fact, notice that there is the following isomorphism:

$$(X \to \mathsf{P_S}Y) = \left(X \to \sum_{x:\mathsf{S}} (x \equiv \top \to Y)\right) \cong \sum_{f:X\to\mathsf{S}} \left(\left(\sum_{x:X} f\,x \equiv \top\right) \to Y\right).$$

An inhabitant of the last type can be considered as a map between a subtype U of X and Y. The subtype U is of the form $\sum_{x:X} f\,x \equiv \top$ for a certain function $f : X \to \mathsf{S}$. The type S behaves like a type of truth values, where $\top$ corresponds to truth and $\bot$ to falsehood. The function f can then be seen as a predicate over X with values in S. In this sense U corresponds to a subtype of X characterized by the predicate f.

The type S is typically called the *Sierpinski set* (Escardó 2004) or *Rosolini's dominance* (Rosolini 1986). It is a fundamental ingredient in the development of synthetic domain theory (Hyland 1990) and synthetic topology (Bauer and Lesnik 2012).

11. Conclusions

In this paper, we studied the question of whether the delay datatype quotiented by weak bisimilarity is still a monad. As we saw, different approaches to quotients in type theory result in different answers. In the quotients-as-setoids approach, the answer is immediately positive. We focused on the more interesting and (as it turned out) more difficult case of the quotient types à la Hofmann. The main issue in this case, as highlighted in Section 5, is that quotienting interacts badly with infinite type formers, such as datatypes of non-wellfounded or non-finitely branching trees; such type formers do not commute with quotienting. For the delay datatype, and more generally for types that can be injectively embedded into streams or countably branching trees, a solution is possible assuming the axiom of countable choice. We also witnessed that essentially the same solution can be

[‡] Notice that, by definition, S has another induction principle given by its dependent eliminator. This induction principle is not strong enough to construct the term p and we need to recourse to the stronger induction principle of the free ωcppo on 1.

employed to prove that the quotiented delay datatype delivers free ωcppos, again relying on countable choice.

We also presented a different monad for partiality in homotopy type theory. Our construction differs from that by Altenkirch et al. (2017) in that we only employ ordinary higher inductive types, while theirs needs higher inductive–inductive types. As a consequence, our construction can be directly implemented in proof assistants such as Coq, which is currently lacking support for inductive–inductive types.

We would also like to see whether it is possible to prove the quotiented delay monad to be the initial complete Elgot monad. Classically, the lifting monad enjoys this characterization (Goncharov et al. 2015). At this stage, it is not clear to us whether the same result can be recovered semi-classically, i.e., without having to subscribe to too non-constructive axioms such as LPO.

Acknowledgements

We thank Thorsten Altenkirch, Andrej Bauer, Nils Anders Danielsson, Bas Spitters, our ICTAC 2015 reviewers, Martín Escardó and MSCS reviewers for comments and suggestions.

Financial Support

This work was supported by the European Regional Development Fund (the Estonian CoE project EXCS, 3.2.0101.08-0013, and the ICT national programme project ‘Coinduction,’ 3.2.1201.13-0029); the Estonian Science Foundation (J.C., N.V., grant no. 9219), (T.U., grant no. 9475); the Estonian Ministry of Education and Research (institutional research grant IUT33-13) and the Engineering and Physical Sciences Research Council (J.C., grant no. EP/M016951/1).

References

Abbott, M., Altenkirch, T. and Ghani, N. (2005). Containers: Constructing strictly positive types. *Theoretical Computer Science* **342** (1) 3–27.

Ahman, D., Chapman, J. and Uustalu, T. (2014). When is a container a comonad? *Logical Methods in Computer Science* **10** (3), article 14.

Altenkirch, T., Danielsson, N.A. and Kraus, N. (2017). Partiality, revisited: The partiality monad as a quotient inductive-inductive type. In: Esparza, J. and Murawski, A. (eds.) *Proceedings of the 20th International Conference on Foundations of Software Science and Computation Structures, FoSSaCS 2017*, Lecture Notes in Computer Science, vol. 10203, Springer, Heidelberg, 534–549.

Bauer, A., Gross, J., Lumsdaine, P.L., Shulman, M., Sozeau, M. and Spitters, B. (2017). The HoTT library: A formalization of homotopy type theory in Coq. In: *Proceedings of 6th ACM SIGPLAN Conference on Certified Programs and Proofs, CPP 2017*, ACM, New York, 164–172.

Bauer, A. and Lesnik, D. (2012). Metric spaces in synthetic topology. *Annals of Pure and Applied Logic* **163** (2) 87–100.

Benton, N., Kennedy, A. and Varming, C. (2009). Some domain theory and denotational semantics in Coq. In: Berghofer, S., Nipkow, T. Urban, C. and Wenzel, M. (eds.) *Proceedings of 22nd*

International Conference on Theorem Proving in Higher Order Logics, TPHOLs 2009, Lecture Notes in Computer Science, vol. 5674, Springer, Heidelberg, 115–130.

Capretta, V. (2005). General recursion via coinductive types. *Logical Methods in Computer Science*, 1(2), article 1.

Chicli, L., Pottier, L. and Simpson, C. (2003). Mathematical quotients and quotient types in Coq. In: Geuvers, H. and Wiedijk, F. (eds.) *Selected Papers from Int. Wksh. on Types for Proofs and Programs, TYPES 2002*, Lecture Notes in Computer Science, vol. 2646, Springer, Heidelberg, 95–107.

Cockett, R., Díaz-Boïls, J., Gallagher, J. and Hrubes, P. (2012). Timed sets, complexity, and computability. In: Berger, U. and Mislove, M. (eds.) *Proceedings of 28th Conference on the Mathematical Foundations of Program Semantics, MFPS XXVIII*, Electronic Notes in Theoretical Computer Science, vol. 286, Elsevier, Amsterdam, 117–137.

Coquand, T., Mannaa, B. and Ruch, F. (2017). Stack semantics of type theory. Preprint. `https://arxiv.org/abs/1701.02571`

Escardó, M. (2004). Synthetic topology of data types and classical spaces. In: Desharnais, J. and Panangaden, P. (eds.) *Proceedings of Wksh. on Domain-Theoretical Methods for Probabilistic Programming*, Electronic Notes in Theoretical Computer Science, vol. 87, Elsevier, Amsterdam, 21–156.

Goncharov, S., Rauch, C. and Schröder, L. (2015). Unguarded recursion on coinductive resumptions. In: Ghica, D. (ed.) *Proceedings of 31st Conference on Mathematical Foundations of Programming Semantics, MFPS XXXI*, Electronic Notes in Theoretical Computer Science, vol. 319, Elsevier, Amsterdam, 183–198.

Hofmann, M. (1997). *Extensional Constructs in Intensional Type Theory. CPHS/BCS Distinguished Dissertations.* Springer, London.

Hyland, J.M.E. (1990). First steps in synthetic domain theory. In: Carboni, A., Pedicchio, M. C. and Rosolini, G. (eds.) *Proceedings of International Conference on Category Theory*, Lecture Notes in Mathematics, vol. 1488, Springer, Heidelberg, 131–156.

Hughes, J. (2000). Generalising monads to arrows. *Science of Computer Programming* **37** (1–3) 67–111.

Jacobs, B., Heunen, C. and Hasuo, I. (2009). Categorical semantics for arrows. *Journal of Functional Programming* **19** (3–4) 403–438.

Kraus, N., Escardó, M., Coquand, T. and Altenkirch, T. (2013). Generalizations of Hedberg's theorem. In: Hasegawa, M. (ed.) *Proceedings of 11th International Conference on Typed Lambda Calculi and Applications, TLCA 2013*, Lecture Notes in Computer Science, vol. 7941, Springer, Heidelberg, 173–188.

Maietti, M.E. (1999). About effective quotients in constructive type theory. In: Altenkirch, T., Naraschewski, W. and Reus, B. (eds.) *Selected Papers from International Wksh. on Types for Proofs and Programs, TYPES '98*, Lecture Notes in Computer Science, vol. 1657, Springer, Heidelberg, 166–178.

Martin-Löf, P. (2006). 100 years of Zermelo's axiom of choice: What was the problem with it? *Computer Journal* **49** (3) 345–350.

Moggi, E. (1991). Notions of computation and monads. *Information and Computation* **93** (1) 55–92.

Mulry, P.S. (1994). Partial map classifiers and partial Cartesian closed categories. *Theoretical Computer Science* **136** (1) 109–123.

Norell, U. (2009). Dependently typed programming in Agda. In: Koopman, P., Plasmeijer, R. and Swierstra, S.D. (eds.) *Revised Lectures from Proceedings of the 6th International School on*

Advanced Functional Programming, AFP 2008, Lecture Notes in Computer Science, vol. 5832, Springer, Heidelberg, 230–266.

Nuo, L. (2015). *Quotient types in type theory*. PhD thesis, University of Nottingham.

Rosolini, G. (1986). *Continuity and Effectiveness in Topoi*. PhD thesis, University of Oxford.

Troelstra, A.S. and Van Dalen, D. (1988). *Constructivism in Mathematics: An Introduction*, vol. I. Studies in Logic and the Foundations of Mathematics, vol. 121, North-Holland, Amsterdam.

The Univalent Foundations Program (2013). *Homotopy Type Theory: Univalent Foundations of Mathematics*. Institute for Advanced Study, Princeton, NY. `http://homotopytypetheory.org/book`.