What Monads Can and Cannot Do with a Bit of Extra Time

Rasmus Ejlers Møgelberg
IT University of Copenhagen, Denmark

Maaike Annebet Zwart
IT University of Copenhagen, Denmark

Abstract

The delay monad provides a way to introduce general recursion in type theory. To write programs that use a wide range of computational effects directly in type theory, we need to combine the delay monad with the monads of these effects. Here we present a first systematic study of such combinations.

We study both the coinductive delay monad and its guarded recursive cousin, giving concrete examples of combining these with well-known computational effects. We also provide general theorems stating which algebraic effects distribute over the delay monad, and which do not. Lastly, we salvage some of the impossible cases by considering distributive laws up to weak bisimilarity.

2012 ACM Subject Classification Theory of computation → Program semantics; Theory of computation → Type structures

Keywords and phrases Delay Monad, Monad Compositions, Distributive Laws, Guarded Recursion, Type Theory

Digital Object Identifier 10.4230/LIPIcs.CSL.2024.39

Related Version An appendix containing the full proofs is available at arXiv.
Extended Version: `http://arxiv.org/abs/2311.15919`

Supplementary Material *Software (Source Code)*: `https://github.com/maaikezwart/Agda-proofs/tree/main/What%20monads%20can%20and%20cannot%20do`
archived at `swh:1:dir:e29201a731f2bffb8137ffa7b0eb09fb388d62d1`

Funding Independent Research Fund Denmark, grant number 2032-00134B.

1 Introduction

Martin Löf type theory [29] is a language that can be understood both as a logic and a programming language. For the logical interpretation it is crucial that all programs terminate. Still, one would like to reason about programming languages with general recursion, or even write general recursive programs inside type theory. One solution to this problem is to encapsulate recursion in a monad, such as the delay monad D. This monad maps an object X to the coinductive solution to $DX \cong X + DX$. The right inclusion into the sum of the above isomorphism introduces a computation step, and infinitely many steps correspond to divergence. Capretta [8] showed how D introduces general recursion via an iteration operator of type $(X \to D(X + Y)) \to X \to DY$. For this reason, D has been used to model recursion in type theory [14, 41, 4], and in particular forms part of the basis of interaction trees [42].

The delay monad has a guarded recursive variant defined using Nakano's [33] fixed point modality $\triangleright$. Data of type $\triangleright X$ should be thought of as data of type X available only one time step from now. This modal operator has a unit $\mathsf{next} : X \to \triangleright X$ transporting data to the future, and a fixed point operator $\mathsf{fix} : (\triangleright X \to X) \to X$ mapping productive definitions to their fixed points satisfying $\mathsf{fix}(f) = f(\mathsf{next}(\mathsf{fix}(f)))$. Guarded recursion can be modelled in the topos of trees [7] – the category $\mathsf{Set}^{\omega^{\mathrm{op}}}$ of presheaves on the ordered natural numbers – by defining $(\triangleright X)(0) = 1$ and $(\triangleright X)(n+1) = X(n)$. The guarded delay monad D^{g} is defined as the free monad on $\triangleright$, i.e., the inductive (and provably also coinductive) solution to $D^{\mathsf{g}}X \cong X + \triangleright(D^{\mathsf{g}}X)$.

The two delay monads can be formally related by moving to a multiclock variant of guarded recursion, in which the modal operator $\triangleright$ is indexed by a clock variable κ, which can be universally quantified. Then by defining the guarded delay monad $D^{\kappa}X$ to be the unique solution to $D^{\kappa}X \cong X + \triangleright^{\kappa}(D^{\kappa}X)$, the coinductive variant can be encoded [2] as $DX \overset{\mathsf{def}}{=} \forall\kappa.D^{\kappa}X$. In this paper we will work informally in Clocked Cubical Type Theory (CCTT) [5], a type theory in which such encodings of coinductive types can be formalised and proven correct.

Unlike the coinductive variant, the guarded delay monad has a fixed point operator of type $((X \to D^{\kappa}Y) \to (X \to D^{\kappa}Y)) \to (X \to D^{\kappa}Y)$, defined using fix. For the coinductive delay monad, fixed points only exist for continuous maps, but in the guarded case, continuity is a consequence of a causality property enforced in types using $\triangleright$. As a consequence, higher order functional programming languages with recursion can be embedded in type theories like CCTT by interpreting function spaces as Kleisli exponentials for D^{κ}. For example, Paviotti et al. [36] showed how to model the simply typed lambda calculus with fixed point terms (PCF), and proved adequacy of the model, all in a type theory with guarded recursion. These results have since been extended to languages with recursive types [31] and (using an impredicative universe) languages with higher-order store [38]. This suggests that the guarded delay monad can be used for programming and reasoning about programs using a wide range of advanced computational effects directly in type theory. However, a mathematical theory describing the interaction of the delay monads with other monads is still lacking, even for basic computational effects.

1.1 Combining the Delay Monad With Other Effects

In this paper, we present a first systematic study of monads combining delay with other effects. We first show how to combine the delay monad with standard monads known from computational effects: exceptions, reader, global state and the selection monad. Most of these follow standard combinations of effects and non-termination known from domain theory, but, the algebraic status of these combinations is simpler than in the domain theoretic case: Whereas the latter can be understood as free monads for order-enriched algebraic theories [20], the combinations with the guarded recursive delay monad are simply free models of theories in the standard sense, with the caveat that the arity of the step operation is non-standard.

The rest of the paper is concerned with distributive laws of the form $TD \to DT$, where T is any monad and D is the delay monad in any of the two forms mentioned above. Such a distributive law distributes the operations of T over steps, and equips the composite DT with a monad structure that describes computations whose other effects are only visible upon termination. This is the natural monad for example in the case of writing to state, when considering non-determinism and observing must-termination, or for computing data contained in data structures such as trees or lists.

There are two natural ways of distributing an n-ary operation op over computation steps: The first is to execute each of the n input computations in sequence until they have all terminated, the second is to execute the n inputs in parallel, delaying terminated computations until all inputs have terminated. We show that sequential execution yields a distributive law for algebraic monads (monads generated by algebraic theories) where all equations are balanced, i.e., the number of occurrences of each variable on either side is the same. Trees, lists, and multisets are examples of such monads.

The requirement of balanced equations is indeed necessary. This was observed already by Møgelberg and Vezzosi [32] who showed that for the finite powerset monad, sequential distribution of steps over the union operator was not well defined, and parallel distribution did not yield a distributive law due to miscounting of steps. Here we strengthen this result to show that no distributive law is possible for the finite powerset monad over the coinductive delay monad. At first sight it might seem that the culprit in this case is the idempotency axiom. However, we show that in some cases it is possible to distribute idempotent operations over the coinductive delay monad, but not over the guarded one (we show this just for commutative operations).

Finally, we show that if one is willing to work up to weak bisimilarity, i.e., equating elements of the delay monad that only differ by a finite number of computation steps, then one can construct a distributive law $TD \to DT$ for any monad T generated by an algebraic theory with no drop equations (equations where a variable appears on one side, but not the other). To make this precise, we formulate this result as a distributive law of monads on a category of setoids.

Agda Formalisation

Some of the results presented in this paper have been formalised in Agda using Vezzosi's Guarded Cubical library[1]. The code can be found at `https://tinyurl.com/WMCDAgda`.

2 Monads and Algebraic Theories

In this background section we briefly remind the reader of algebraic theories and free model monads for an algebraic theory. We mention different classes of equations that play a role in our analysis of monad compositions, and we discuss distributive laws for composing monads.

▶ **Definition 1** (Algebraic Theory). *An algebraic theory A consists of a signature Σ_A and a set of equations E_A. The signature is a set of operation symbols with arities given by natural numbers. The signature Σ_A together with a set of variables X inductively defines the set of A-terms: Every variable $x : X$ is a term, and for each operation symbol op in Σ_A, if op has arity n and $t_1, \dots, t_n$ are terms, then $\mathrm{op}(t_1, \dots, t_n)$ is a term. The set of equations contains pairs of terms (s, t) in a finite variable context which are to be considered equal, often written as $s = t$. These pairs then define an equivalence relation on terms via equational logic.*

▶ **Example 2** (Monoids). The algebraic theory of monoids has a signature consisting of a constant c and a binary operation $*$, satisfying the left and right unital equations: $\forall x : X.\, c * x = x$ and $\forall x : X.\, x * c = x$, and associativity: $\forall x, y, z : X.\, (x * y) * z = x * (y * z)$. Commutative monoids also include the commutativity equation: $\forall x, y : X.\, x * y = y * x$.

▶ **Definition 3** (Category of Models). *A model of an algebraic theory (Σ_A, E_A) is a set X together with an interpretation $\mathrm{op}_X : X^n \to X$ of each n-ary operation op in Σ_A, such that $s_X = t_X$ for each equation $s = t$ in E_A. Here s_X is the interpretation of s defined inductively using the interpretation of operations.*

A homomorphism between two models $(X, (-)_X)$ and $(Y, (-)_Y)$ is a morphism $h : X \to Y$ such that $h(\mathrm{op}_X(x_1, \dots, x_n)) = \mathrm{op}_Y(h(x_1), \dots, h(x_n))$ for each n-ary operation op in Σ_A and $x_1, \dots, x_n : X$. The models of an algebraic theory and homomorphisms between them form a category called the category of algebras *of the algebraic theory, denoted A-alg.*

[1] `https://github.com/agda/guarded`

The free model of an algebraic theory A with variables in a set X consists of the set of equivalence classes of A-terms in context X. The functor $F : \mathbf{Set} \to A\text{-alg}$ sending each set X to the free model of A on X is left adjoint to the forgetful functor sending each A-model to its underlying set. This adjunction induces a monad on **Set**, called the *free model monad* of the algebraic theory [24, 23, 25]. The category of algebras of A is isomorphic to the Eilenberg-Moore category of this monad. If a monad T is isomorphic to the free model monad of an algebraic theory, then we say that T *is presented* by that algebraic theory.

▶ **Example 4** (Boom Hierarchy Monads [30])**.** The binary tree monad, the list monad, the multiset monad and the powerset monad are the free model monads of respectively the theories of:

- Magmas, the theory consisting of a constant and a binary operation satisfying the left and right unit equations.
- Monoids, which are magmas satisfying the associativity equation.
- Commutative monoids.
- Idempotent commutative monoids, which are also known as semilattices.

The equations of an algebraic theory determine much of the behaviour of its free model monad. For example, linear equations result in monads that always compose with commutative monads [26, 34]. In this paper, we built upon the ideas of Gautam [16] and Parlant et al [35], and distinguish the following classes of equations:

▶ **Definition 5.** *Write* $\mathrm{var}(t)$ *for the set of variables that appear in a term* t*. We say that an equation* $s = t$ *is:*

Linear *if* $\mathrm{var}(s) = \mathrm{var}(t)$ *and each variable in these sets appears exactly once in both* s *and* t*. Example:* $x * y = y * x$*.*

Balanced *if* $\mathrm{var}(s) = \mathrm{var}(t)$ *and each variable in these sets appears equally many times in* s *and* t*. Example:* $(x * y) * (y * z) = (y * y) * (x * z)$*.*

Dup *if there is an* $x : \mathrm{var}(s) \cup \mathrm{var}(t)$*, such that* x *appears* ≥ 2 *times in* s *and/or* t*. Example: the balanced equation above, as well as* $x * x = x * x * x$ *and* $x \wedge (x \vee y) = x$*.*

Drop *if* $\mathrm{var}(s) \neq \mathrm{var}(t)$*. Example:* $x \wedge (x \vee y) = x$*.*

▶ Remark 6. Notice that these types of equations are not mutually exclusive. An equation can for instance be both dup and drop, such as the absorption equation $x \wedge (x \vee y) = x$.

2.1 Distributive Laws

One way of composing two monads is via a *distributive law* describing the interaction between the two monads [6].

▶ **Definition 7** (Distributive Law)**.** *Given monads* $\langle S, \eta^S, \mu^S \rangle$ *and* $\langle T, \eta^T, \mu^T \rangle$*, a distributive law distributing* S *over* T *is a natural transformation* $\zeta : ST \to TS$ *satisfying the following axioms:*

$$\zeta \circ \eta^S T = T\eta^S \qquad \zeta \circ S\eta^T = \eta^T S \qquad \text{(unit axioms)}$$

$$\zeta \circ \mu^S T = T\mu^S \circ \zeta S \circ S\zeta \qquad \zeta \circ S\mu^T = \mu^T S \circ T\zeta \circ \zeta T \qquad \text{(multiplication axioms)}$$

▶ **Example 8** (Lists and Multisets)**.** Distributive laws are named after the well-known distributivity of multiplication over addition: $a * (b + c) = (a * b) + (a * c)$. Many distributive laws follow the same distribution pattern. For example, the list monad distributes over the multiset monad in this way: $\zeta[\wr a \int, \wr b, c \int] = \wr [a, b], [a, c] \int$. However, this is by no means the only way a distributive law can function.

▶ **Theorem 9** (Beck [6]). *Let $\mathcal{C}$ be a category, and $\langle S, \eta^S, \mu^S \rangle$ and $\langle T, \eta^T, \mu^T \rangle$ two monads on $\mathcal{C}$. If $\zeta : ST \to TS$ is a distributive law, then the functor TS carries a monad structure with unit $\eta^T \eta^S$ and multiplication $\mu^T \mu^S \circ T\zeta S$.*

We frequently use the following equivalence in our proofs:

▶ **Theorem 10** (Beck [6]). *Given two monads $\langle S, \eta^S, \mu^S \rangle$ and $\langle T, \eta^T, \mu^T \rangle$ on a category $\mathcal{C}$, there is a bijective correspondence between distributive laws of type $ST \to TS$, and liftings of T to the Eilenberg-Moore category $\mathcal{C}^S$ of S.*

Here, a lifting of T to $\mathcal{C}^S$ is an assignment mapping S-algebra structures on a set X to S-algebra structures on TX such that η^T and μ^T are S-algebra homomorphisms.

3 Guarded Recursion and the Delay Monad

In this paper we work informally in Clocked Cubical Type Theory (CCTT) [5]. At present, this is the only known theory combining the features we need: Multiclocked guarded recursion and quotient types (to express free monads). Here we remind the reader of the basic principles of CCTT, but we refer to Kristensen et al. [5] for the full details, including a denotational semantics for CCTT.

3.1 Algebraic Theories in Cubical Type Theory

CCTT is an extension of Cubical Type Theory (CTT) [12], which in turn is a version of Homotopy Type Theory (HoTT) [39] that gives computational content to the univalence axiom. In CTT, the identity type of Martin-Löf type theory is replaced by a path type, which we shall write infix as $t =_A u$, often omitting the type A of t and u. We will work informally with $=$, using its standard properties such as function extensionality.

A type A is a *homotopy proposition* (or hprop) in HoTT and CTT, if any two elements of A are equal, and an *hset* if $x =_A y$ is an hprop for all $x, y : A$. Assuming a universe of small types, one can encode universes hProp and hSet of homotopy sets and propositions in the standard way. The benefit of working with hsets is that there is no higher structure to consider. In particular, the collection of hsets and maps between these forms a category in the sense of HoTT [39], and so basic category theoretic notions such as functors and monads on hsets can be formulated in the standard way.

The notion of algebraic theory can also be read directly in CTT this way. Moreover, the free monads on algebraic theories can be defined using higher inductive types (HITs). These are types given inductively by constructors for terms as well as for equalities. For example, the format for HITs used in CCTT [5] (adapted from Cavallo and Harper [9]) is expressive enough[2].

We write type equivalence as $A \simeq B$. For hsets this just means that there are maps $f : A \to B$ and $g : B \to A$ that are inverses of each other, as expressed in CTT using path equality.

3.2 Multi-Clocked Guarded Recursion

CCTT extends CTT with multi-clock guarded recursion. The central component in this is a modal type operator $\triangleright$ indexed by clocks κ, used to classify data that is delayed by one time step on clock κ. The most important typing rules of CCTT are collected in Figure 1.

[2] Full details are in the appendix in the extended version on ArXiv

$$\frac{\Gamma \vdash}{\Gamma, \kappa : \mathsf{clock} \vdash} \qquad \frac{\kappa : \mathsf{clock} \in \Gamma}{\Gamma, \alpha : \kappa \vdash} \qquad \frac{\Gamma, \mathsf{TimeLess}(\Gamma') \vdash t : \triangleright(\alpha{:}\kappa).A \qquad \Gamma, \beta : \kappa, \Gamma' \vdash}{\Gamma, \beta : \kappa, \Gamma' \vdash t\,[\beta] : A\,[\beta/\alpha]}$$

$$\frac{\Gamma, \alpha : \kappa \vdash t : A}{\Gamma \vdash \lambda(\alpha{:}\kappa).t : \triangleright(\alpha{:}\kappa).A} \qquad \frac{\Gamma, \kappa : \mathsf{clock} \vdash t : A}{\Gamma \vdash \Lambda\kappa.t : \forall\kappa.A} \qquad \frac{\Gamma \vdash t : \forall\kappa.A \qquad \Gamma \vdash \kappa' : \mathsf{clock}}{\Gamma \vdash t[\kappa'] : A[\kappa'/\kappa]}$$

$$\frac{\Gamma \vdash t : \triangleright^\kappa A \to A}{\Gamma \vdash \mathsf{dfix}^\kappa\, t : \triangleright^\kappa A} \qquad \frac{\Gamma \vdash t : \triangleright^\kappa A \to A}{\Gamma \vdash \mathsf{pfix}^\kappa\, t : \triangleright(\alpha{:}\kappa).(\mathsf{dfix}^\kappa t)\,[\alpha] =_A t(\mathsf{dfix}^\kappa t)}$$

Figure 1 Selected typing rules for Clocked Cubical Type Theory [5]. The telescope $\mathsf{TimeLess}(\Gamma')$ is composed of the timeless assumptions in Γ, i.e. interval variables and faces (as in Cubical Type Theory) as well as clock variables.

Clocks are introduced as special assumptions $\kappa : \mathsf{clock}$ in a context, and can be abstracted and applied to terms of the type $\forall\kappa.A$ which behaves much like a Π-type for clocks. Like function extensionality, extensionality for $\forall\kappa.A$ also holds in CCTT.

The rules for $\triangleright$ also resemble those of Π-types: Introduction is by abstracting special assumptions $\alpha : \kappa$ called *ticks* on the clock κ. Since ticks can appear in terms, the modal type $\triangleright(\alpha{:}\kappa).A$ binds α in A, just like a Π-type binds a variable. We write $\triangleright^\kappa A$ for $\triangleright(\alpha{:}\kappa).A$ when α does not appear in A. The introduction rule for $\triangleright$ can be read as stating that if t has type A after the tick α, then $\lambda(\alpha{:}\kappa).t$ has type $\triangleright(\alpha{:}\kappa).A$ now.

The modality $\triangleright$ is eliminated by applying a term to a tick. Note that the term t applied to the tick β cannot already contain β freely. This restriction prevents t from being applied twice to the same tick, which would construct terms of type $\triangleright^\kappa \triangleright^\kappa A \to \triangleright^\kappa A$, collapsing two steps into one. Moreover, t cannot contain any variables nor other ticks occurring in the context after β, only *timeless* assumptions, i.e., clocks, interval assumptions and faces. One application of timeless assumptions is to type the extensionality principle for $\triangleright$:

$$(t =_{\triangleright(\alpha:\kappa).A} u) \simeq \triangleright(\alpha{:}\kappa).(t\,[\alpha] =_A u\,[\alpha]). \tag{1}$$

For all explicit applications of terms to ticks in this paper, the term will not use timeless assumptions. The usual η and β laws hold for tick abstraction and application.

The use of ticks for programming with $\triangleright$ implies that $\triangleright$ is an applicative functor, and can even be given a dependent applicative action of type

$$\Pi(f : \triangleright^\kappa(\Pi(X : A).B(x)).\Pi(y : \triangleright^\kappa A).\triangleright(\alpha{:}\kappa).B(y\,[\alpha]).$$

Ticks are named in CCTT for reasons of normalisation [3], but are essentially identical. This is expressed in type theory as the *tick irrelevance principle*:

$$\mathsf{tirr}^\kappa : \Pi(x : \triangleright^\kappa A).\triangleright(\alpha{:}\kappa).\triangleright(\beta{:}\kappa).(x\,[\alpha] =_A x\,[\beta]). \tag{2}$$

The term tirr^κ is defined in CCTT using special combinators on ticks, allowing for computational content to tirr^κ. This means that the rule for tick application is more general than the one given in Figure 1. However, we will not need this further generality for anything apart from tirr^κ, which we use directly.

Finally, CCTT has a fixed point operator dfix which unfolds up to path equality as witnessed by pfix. Using these, one can define $\mathsf{fix}^\kappa : (\triangleright^\kappa A \to A) \to A$ as $\mathsf{fix}^\kappa(t) = t(\mathsf{dfix}^\kappa t)$ and prove $\mathsf{fix}^\kappa(t) = t(\mathrm{next}^\kappa(\mathsf{fix}^\kappa(t)))$ where $\mathrm{next}^\kappa \stackrel{\mathsf{def}}{=} (\lambda(x : A).\lambda(\alpha{:}\kappa).x) : A \to \triangleright^\kappa A$. Note that this uses that also variables appearing before a tick in a context can be introduced. This is not the case in all Fitch-style modal type theories [11, 17].

3.3 Guarded Recursive Types

A guarded recursive type is a recursive type in which the recursive occurrences of the type are all guarded by a $\triangleright$. These can be encoded up to equivalence of types using fixed points of maps on the universe. Our primary example is the guarded recursive delay monad D^κ defined to map an X to the recursive type

$$D^\kappa X \simeq X + \triangleright^\kappa(D^\kappa X).$$

We write $\mathrm{now} : X \to D^\kappa X$ and $\mathrm{step} : \triangleright^\kappa(D^\kappa X) \to D^\kappa X$ for the two maps given by inclusion and the equivalence above.

Since $\triangleright^\kappa$ preserves the property of being an hset, one can prove by guarded recursion that $D^\kappa X$ is an hset whenever X is. D^κ can be seen as a free construction in the following sense.

▶ **Definition 11.** *A* delay algebra *on the clock κ is an hset X together with a map $\triangleright^\kappa X \to X$.*

Given an hset X, the hset $D^\kappa X$ carries a delay algebra structure. It is the free delay algebra in the sense that given any other delay algebra (Y, ξ), and a map $f : X \to Y$, there is a unique homomorphism $\overline{f} : D^\kappa X \to Y$ extending f along now, defined by the clause

$$\overline{f}(\mathrm{step}(x)) = \xi(\lambda(\alpha\!:\!\kappa).\overline{f}(x\,[\alpha])). \tag{3}$$

This is a recursive definition that can be encoded as a fixed point of a map $h : \triangleright^\kappa(D^\kappa X \to Y) \to (D^\kappa X \to Y)$ defined using the clause $h(g)(\mathrm{step}(x)) = \xi(\lambda(\alpha\!:\!\kappa).(g\,[\alpha])(x\,[\alpha]))$. In this paper we use the simpler notation of Equation (3) for such definitions rather than the explicit use of fix^κ.

We sketch the proof that $\overline{f}$ is the unique homomorphism extending f, to illustrate the use of fix^κ for proofs. Suppose g is another such extension. To use guarded recursion, assume that $\triangleright^\kappa(g = \overline{f})$. We show that $g(\mathrm{step}(x)) = \xi(\lambda(\alpha\!:\!\kappa).\overline{f}(x\,[\alpha]))$. Since g is a homomorphism: $g(\mathrm{step}(x)) = \xi(\lambda(\alpha : \kappa).g(x\,[\alpha]))$. So by extensionality for $\triangleright$ (1) it suffices to show that $\triangleright(\alpha\!:\!\kappa).(g(x\,[\alpha]) = \overline{f}(x\,[\alpha]))$, which follows from the guarded recursion hypothesis.

Tick irrelevance implies that D^κ is a commutative monad in the sense of Kock [22].

3.4 Encoding Coinductive Types

Coinductive types can be encoded using a combination of guarded recursive types and quantification over clocks. This was first observed by Atkey and McBride [2]. We recall the following special case of a more general theorem for this in CCTT [5]. First a definition.

▶ **Definition 12.** *A functor $F : \mathsf{hSet} \to \mathsf{hSet}$* commutes with clock quantification*, if the canonical map $F(\forall\kappa.(X\,[\kappa])) \to \forall\kappa.F(X\,[\kappa])$ is an equivalence for all $X : \forall\kappa.\mathsf{hSet}$. An hset X is* clock irrelevant *if the constant functor to X commutes with clock quantification, i.e. if the canonical map $X \to \forall\kappa.X$ is an equivalence.*

Note that functors commuting with clock quantification map clock irrelevant types to clock irrelevant types.

▶ **Theorem 13** ([5]). *Let F be an endofunctor on the category of hsets commuting with clock quantification, and let $\nu^\kappa F$ be the guarded recursive type satisfying $F(\triangleright^\kappa(\nu^\kappa F)) \simeq \nu^\kappa F$, then $\nu F \overset{\mathit{def}}{=} \forall\kappa.\nu^\kappa F$ carries a final coalgebra structure for F.*

In order to apply Theorem 13, of course, one needs a large collection of functors F commuting with clock quantification. Fortunately, the collection of such functors is closed under almost all type constructors, including finite sum and product, Π and Σ types, $\triangleright$, $\forall\kappa$, and guarded recursive types [5, Lemma 4.2]. Clock irrelevant types are likewise closed under the same type constructors, and path equality. The only exception to clock irrelevance is the universe type.

For example, if X is clock irrelevant, then $F(Y) = X + Y$ commutes with clock quantification, and so $DX \stackrel{\mathsf{def}}{=} \forall\kappa.D^{\kappa}X$ is the coinductive solution to $DX \simeq X + DX$.

CCTT moreover has a principle of *induction under clocks* allowing one to prove that many HITs are clock irrelevant, including the empty type, booleans and natural numbers. Moreover, one can prove the following.

▶ **Proposition 14.** *Let $A = (\Sigma_A, E_A)$ be an equational theory such that Σ_A and E_A are clock irrelevant. Then the free model monad T commutes with clock quantification. In particular, $T(X)$ is clock irrelevant for all clock irrelevant X.*

The collection of clock irrelevant propositions can be shown to be closed under standard logical connectives. Alternatively, one can assume a global clock constant κ_0, which then can be used to prove that all propositions are clock irrelevant.

▶ **Convention 15.** *In the remainder of this paper, the word* set *will refer to a clock-irrelevant hset, and the word* proposition *will refer to clock-irrelevant homotopy propositions. We will write* Set *and* Prop *for the universes of these. Similarly, whenever we mention functors these are assumed to commute with clock quantification.*

4 Specific Combinations with Delay

In this section we look at some specific examples of monads, and see how they combine with the delay monads. In particular, we will look at the exception, reader, state, and selection monads. Intuitively, these monads model (parts of) the process: read input - compute - do something with the output. For instance, the state monad reads a state, then both updates the current state and gives an output. Combining the state monad with the delay monads allows us to model the fact that the computation in between reading the input and giving the output takes time, and might not terminate.

The examples we give follow the same pattern as the adaptation of these monads to domain theory: we insert a delay monad where one would use lifting in the domain theoretic case. However, we also show that the algebraic status of these monads is much simpler in the guarded recursive case than in the domain theoretic one: they can simply be understood as being generated by algebraic theories where one operation (step) has a non-standard arity. In the domain theoretic case, the algebraic description is in terms of enriched Lawvere theories [20]. We give no algebraic description of the combinations with the coinductive delay monads, because this does not by itself have an algebraic description.

First note that for combinations with delay via a distributive law, it is enough to find a distributive law for the guarded recursive version D^{κ}.

▶ **Lemma 16.** *Let T be a monad. A distributive law $\zeta_X : \forall\kappa.T(D^{\kappa}(X)) \to D^{\kappa}(T(X))$ for the guarded delay monad induces a distributive law $TD \to DT$ for the coinductive delay monad. Similarly, if T^{κ} is a family of monads indexed by κ then $T(X) = \forall\kappa.T^{\kappa}(X)$ carries a monad structure.*

Proof. The distributive law can be constructed as the composite

$$T(\forall\kappa.D^{\kappa}(X)) \to (\forall\kappa.T(D^{\kappa}(X))) \to \forall\kappa.D^{\kappa}(T(X)),$$

where κ is fresh for T and X. For the second statement define the multiplication as the composite $\forall\kappa.T^{\kappa}(\forall\kappa.T^{\kappa}(X)) \to (\forall\kappa.T^{\kappa}(T^{\kappa}(X))) \to \forall\kappa.T^{\kappa}(X)$. ◀

Exceptions

The first monad we consider is the exception monad. For a set of exceptions E, the exception monad is given by the functor $(- + E)$, with obvious unit and multiplication. The exception monad is the free model monad of the algebraic theory consisting of a signature with a constant e for each exception in E, and no equations.

It is well known that the exception monad distributes over any monad, and therefore we have a distributive law $\zeta : (D^\kappa(-) + E) \to D^\kappa(- + E)$. The resulting composite monad $D^\kappa(- + E)$ is the free model monad of the theory consisting of constants $e : E$ and a step-operator forming a delay algebra, with no additional equations.

Reading

The reader monad $(-)^R$ is presented by the algebraic theory consisting of a single operation $\mathrm{lookup} : X^R \to X$, satisfying the equations

$$\forall x : X.\ \mathrm{lookup}(\lambda r.x) = x \qquad \forall g : (X^R)^R.\ \mathrm{lookup}(\mathrm{lookup} \circ g) = \mathrm{lookup}(\lambda s.g\, s\, s).$$

To combine the reader monad with the delay monad, we define a distributive law $D^\kappa R \to RD^\kappa$ by the clauses $\zeta(\mathrm{now}\, f) = \lambda r.\, \mathrm{now}(fr)$ and $\zeta(\mathrm{step}\, d) = \lambda r.\, \mathrm{step}(\lambda(\alpha : \kappa).(\zeta(d\,[\alpha]))r)$, where $f : X^R$ and $d : \triangleright^\kappa(X^R)$. The resulting composite monad RD^κ is the free model monad of the theory consisting of lookup and step satisfying the above equations for lookup and

$$\forall d : \triangleright^\kappa(X^R).\ \mathrm{step}(\lambda(\alpha{:}\kappa).\, \mathrm{lookup}(d\,[\alpha])) = \mathrm{lookup}(\lambda r.\, \mathrm{step}(\lambda(\alpha{:}\kappa).d\,[\alpha]\, r)). \tag{4}$$

Global State

Plotkin and Power [37] show that the global state monad $(S \times -)^S$ can be described algebraically by two operations: $\mathrm{lookup} : X^S \to X$ and $\mathrm{update} : X \to X^S$, satisfying four interaction diagrams. They call the category of such algebras *GS-algebras*.

The natural combination of global state and D^κ is $(D^\kappa(S \times -))^S$ describing computations whose steps occur between reading the initial state and writing back the updated state. To describe this monad algebraically define a GSD-algebra to be a GS-algebra which also carries a delay algebra structure satisfying (4) and

$$\forall x : \triangleright^\kappa X.\ \lambda s.\, \mathrm{update}(\mathrm{step}\, x)s = \lambda s.\, \mathrm{step}(\lambda(\alpha{:}\kappa).\, \mathrm{update}(x\,[\alpha])s).$$

Diagrammatically:

$$\begin{tikzcd}
\triangleright^\kappa(X^S) \arrow[r] \arrow[d, "\triangleright^\kappa(\mathrm{lookup})"] & (\triangleright^\kappa X)^S \arrow[r, "\mathrm{step}^S"] & X^S \arrow[d, "\mathrm{lookup}"] \\
\triangleright^\kappa X \arrow[rr, "\mathrm{step}"] & & X
\end{tikzcd}
\qquad
\begin{tikzcd}
\triangleright^\kappa X \arrow[r, "\triangleright^\kappa(\mathrm{update})"] \arrow[d, "\mathrm{step}"] & \triangleright^\kappa(X^S) \arrow[r] & (\triangleright^\kappa X)^S \arrow[d, "\mathrm{step}^S"] \\
X \arrow[rr, "\mathrm{update}"] & & X^S
\end{tikzcd}$$

▶ **Theorem 17.** *The monad $(D^\kappa(S \times -))^S$ is the free model monad of the theory of GSD-algebras.*

Note that also $(D(S \times -))^S$ is a monad by Lemma 16, since the assumption of S being clock irrelevant implies $(D(S \times -))^S \simeq \forall\kappa.((D^\kappa(S \times -))^S)$.

Selecting

The selection monad $\mathcal{J}X = (X \to S) \to X$ takes a function $X \to S$, and selects an input $x : X$ to return [15]. This could, for example, be an input for which the function attains an optimal value. It is a monad similar to the reader monad, with a more advanced input. It is also a close companion to the continuation monad $(X \to S) \to S$, and it has many applications in for example game theory and functional programming [18].

The selection monad combines with the delay monad via a distributive law of type $D^\kappa \mathcal{J} \to \mathcal{J} D^\kappa$. Intuitively, it first gathers all the data from the function $X \to S$, and then computes which element from X to select. This computation takes time and might not terminate. We assume that the initial input is readily available, even though the resulting type of the monad composition is $(D^\kappa X \to S) \to D^\kappa X$. This fact is reflected in the definition of the distributive law below.

The distributive law is similar to the distributive law for the delay monad over the reader monad, and is given by:

$$\zeta(\mathrm{step}(d)) = \lambda g.\, \mathrm{step}(\lambda(\alpha\!:\!\kappa).(\zeta(d\,[\alpha]))g),$$

where $f : (X \to S) \to X$ and $d : \triangleright^\kappa(D^\kappa((X \to S) \to X))$. As a result, both $\mathcal{J}D^\kappa$ and $\mathcal{J}D$ can be equipped with monad structures.

Free Combinations With Delay

The sum of two monads T and S is a monad $T \oplus S$ whose algebras are objects X with algebra structures for both T and S [19]. In terms of algebraic theories, the sum can be understood as combining two theories with no equations between them. The sum of D^κ with any other monad always exists [19, Theorem 4]:

▶ **Corollary 18.** *Let T be a monad, and define $T \oplus D^\kappa$ as the guarded recursive type:*

$$(T \oplus D^\kappa)X \simeq T(X + \triangleright^\kappa((T \oplus D^\kappa)X)).$$

Then $(T \oplus D^\kappa)(X)$ is the carrier of the free T-algebra and delay-algebra structure.

The monad mapping X to $\forall\kappa.(T \oplus D^\kappa)(X)$ includes the coinductive delay monad and T, but we have not been able to prove a general universal property for this. We believe that it is not the sum of the two.

5 Parallel and Sequential Distribution of Operations

We now consider distributive laws of type $TD \to DT$, where D is one of the delay monads and T is any presentable monad. Such laws equip the composite DT with a monad structure, which is the natural one in particular for monads describing data structures, such as those in the Boom hierarchy.

We again focus on distributive laws involving the guarded version of the delay monad, invoking Lemma 16. Intuitively, such a distributive law pulls all the steps out of the algebraic structure of T: it turns a T-structure with delayed elements into a delayed T-structure. There are two obvious candidates for such a lifting: *parallel* and *sequential* computation. We define both of these on operations using guarded recursion. A lifting of terms then follows inductively from lifting each operation in the signature of the presentation of T.

▶ **Definition 19** (Parallel Lifting of Operators)**.** *Let A be an algebraic theory, and let X be an A-model. Define, for each n-ary operation* op *in A, a lifting* $\mathrm{op}^{par}_{D^\kappa X} : (D^\kappa X)^n \to D^\kappa X$ *by:*

$$\begin{aligned}\mathrm{op}^{par}_{D^\kappa X}(\mathrm{now}\, x_1, \ldots, \mathrm{now}\, x_n) &= \mathrm{now}(\mathrm{op}_X(x_1, \ldots, x_n))\\ \mathrm{op}^{par}_{D^\kappa X}(x_1, \ldots, x_n) &= \mathrm{step}(\lambda\alpha.(\mathrm{op}^{par}_{D^\kappa X}(x'_1, \ldots, x'_n))),\end{aligned}$$

where the second clause only applies if one of the x_i is of the form $\mathrm{step}(x''_i)$ *and*

$$x'_i = \begin{cases} x_i & \text{if } x_i = \mathrm{now}(x''_i)\\ x''_i\,[\alpha] & \text{if } x_i = \mathrm{step}(x''_i)\end{cases}$$

▶ **Definition 20** (Sequential Lifting of Operators)**.** *Let A be an algebraic theory, and let X be an A-model. Define, for each n-ary operation* op *in A, a lifting* $\mathrm{op}^{seq}_{D^\kappa X} : (D^\kappa X)^n \to D^\kappa X$ *by:*

$$\begin{aligned}\mathrm{op}^{seq}_{D^\kappa X}(\mathrm{now}\, x_1, \ldots, \mathrm{now}\, x_n) &= \mathrm{now}(\mathrm{op}_X(x_1, \ldots, x_n))\\ \mathrm{op}^{seq}_{D^\kappa X}(\mathrm{now}\, x_1, \ldots, \mathrm{step}\, x_i, \ldots x_n) &= \mathrm{step}(\lambda\alpha.(\mathrm{op}^{seq}_{D^\kappa X}(\mathrm{now}\, x_1, \ldots, (x_i\,[\alpha]), \ldots, x_n))),\end{aligned}$$

where, in the second clause, the ith argument is the first not of the form $\mathrm{now}(x'_k)$*.*

In general, for an n-ary operation op, parallel lifting evaluates all arguments of the form $\mathrm{step}(x_i)$ in parallel, and sequential lifting evaluates them one by one from the left. Parallel lifting of an operator therefore terminates in as many steps as the maximum required for each of its inputs to terminate, while sequential lifting terminates in the sum of the number of steps required for each input.

The evaluation order of arguments in the case of sequential lifting is inessential, which can be proved using guarded recursion and tick irrelevance.

▶ **Lemma 21.** *Let A be an algebraic theory, and let* op *be an n-ary operation in A. Then*

$$\mathrm{op}^{seq}_{D^\kappa X}(x_1, \ldots, \mathrm{step}(x_i), \ldots x_n) = \mathrm{step}(\lambda(\alpha\!:\!\kappa).\mathrm{op}^{seq}_{D^\kappa X}(x_1, \ldots, x_i\,[\alpha], \ldots, x_n)).$$

5.1 Preservation of Equations

Parallel lifting preserves all non-drop equations, whereas sequential lifting only preserves balanced equations. We prove this in the two following propositions. We write $s^{par}_{D^\kappa X}$ for the interpretation of a term s on $D^\kappa X$ defined by induction of s using the parallel lifting of operations, and likewise $s^{seq}_{D^\kappa X}$ for the interpretation defined using sequential lifting of operations.

▶ **Proposition 22** (Parallel Preserves Non-Drop)**.** *Let $A = (\Sigma_A, E_A)$ be an algebraic theory, X an A-model, and $s = t$ a non-drop equation that is valid in A. Then also $s^{par}_{D^\kappa X} = t^{par}_{D^\kappa X}$.*

The restriction to non-drop equations is necessary, because divergence in a dropped variable leads to divergence on one side of the equation, but not on the other.

Møgelberg and Vezzosi [32] observed that parallel lifting does not define a distributive law in the case of the finite powerset monad. Their proof uses idempotency, but in fact parallel lifting does not define a monad even just in the presence of a single binary operation.

▶ **Theorem 23.** *Let T be an algebraic monad with a binary operation* op*. Then the natural transformation $\zeta : TD \to DT$ induced by parallel lifting does not define a distributive law, because it fails the second multiplication axiom.*

Proof. The counter example is the same as used by Møgelberg and Vezzosi:

$$\mathrm{op}^{par}_{DDX}(\mu^D(\mathrm{now}(\mathrm{step}\,\mathrm{now}\,x)), \mu^D(\mathrm{step}(\mathrm{now}(\mathrm{now}\,y))) = \mathrm{step}(\mathrm{now}(\mathrm{op}_X(x,y)))$$
$$\mu^D(\mathrm{op}^{par}_{DDX}(\mathrm{now}(\mathrm{step}(\mathrm{now}\,x)), \mathrm{step}(\mathrm{now}(\mathrm{now}\,y)))) = \mathrm{step}(\mathrm{step}(\mathrm{now}(\mathrm{op}_X(x,y)))).$$ ◀

Note that we used the coinductive version of the delay monad in the above theorem. By Lemma 16, this implies the same result for the guarded recursive version.

▶ **Proposition 24** (Sequential Preserves Balanced)**.** *Let A be an algebraic theory, and s, t be two A-terms such that $s = t$ is a balanced equation that is valid in A. Then also $s^{seq}_{D^\kappa X} = t^{seq}_{D^\kappa X}$.*

Balance is necessary. For example, if t and s are terms in a single variable which occurs twice in t and once in s, then $t^{seq}_{D^\kappa X}(\mathrm{step}(x))$ takes at least two steps, but $s^{seq}_{D^\kappa X}(\mathrm{step}(x))$ might take only one. Building on Proposition 24, one can prove the following.

▶ **Theorem 25.** *Let T be the free model monad of algebraic theory $\mathbb{T} = (\Sigma_\mathbb{T}, E_\mathbb{T})$, such that $E_\mathbb{T}$ only contains* balanced *equations. Then sequential lifting defines a distributive law $TD^\kappa \to D^\kappa T$.*

Combining this with Lemma 16 we obtain a distributive law $TD \to DT$ for all T as in Theorem 25.

▶ Remark 26. Since D^κ is a commutative monad, we already know from Manes and Mulry [26] and Parlant [34] that there is a distributive law in the case where $\mathbb{T}$ only has *linear* equations. We can extend this linearity requirement here to allow duplications of variables, as long as there are equally many duplicates on either side of each equation.

▶ **Example 27.** The sequential distributive law successfully combines the delay monad with the binary tree monad, the list monad, and the multiset monad, resulting in the monads $D^\kappa B$, $D^\kappa L$, and $D^\kappa M$, respectively.

6 Idempotent Equations

This section studies distributive laws $TD \to DT$ for T an algebraic monad with an idempotent binary operation "op". Since idempotency is not a balanced equation, as remarked after Proposition 24, sequential distribution does not respect it, and so neither parallel nor sequential distribution define distributive laws in this case. Idempotency turns out to be a tricky equation: We first show an example of such a theory T where no distributive law $TD \to DT$ is possible, then a theory where it is, and finally we show that no distributive law of type $TD^\kappa \to D^\kappa T$ is possible. First observe the following.

▶ **Lemma 28.** *Let T be an algebraic monad with an idempotent binary operation* op *and let $\zeta : TD \to DT$ be a distributive law. There exist binary T-operations* op_1 *and* op' *such that for any T-model X, the lifting of* op *to DX satisfies* $\mathrm{op}(\mathrm{step}\,x, \mathrm{step}\,y) = \mathrm{step}(\mathrm{op}_1(x,y))$ *and either 1)* $\mathrm{op}(\mathrm{step}\,x, y) = \mathrm{step}(\mathrm{op'}(x,y))$ *and* $\mathrm{op'}(x, \mathrm{step}\,y) = \mathrm{op}_1(x,y)$ *or 2)* $\mathrm{op}(\mathrm{step}\,x, y) = \mathrm{op'}(x,y)$ *and* $\mathrm{op'}(x, \mathrm{step}\,y) = \mathrm{step}(\mathrm{op}_1(x,y))$.

Proof Sketch. We just sketch the proof of existence of op_1. Consider the naturality diagram for the unique map $! : 2 \to 1$ from the 2-element set $\{\mathsf{tt}, \mathsf{ff}\}$ to the singleton set $\{\star\}$.

$$\begin{tikzcd}
D(T(2)) \times D(T(2)) \arrow[r, "\mathrm{op}"] \arrow[d, "D(T(!))\times D(T(!))"'] & D(T(2)) \arrow[d, "D(T(!))"] \\
D(T(1)) \times D(T(1)) \arrow[r, "\mathrm{op}"] & D(T(1))
\end{tikzcd}$$

By idempotency, the lower composition maps $(\mathrm{step}(\mathrm{now}(\eta^T(\mathsf{tt}))), \mathrm{step}(\mathrm{now}(\eta^T(\mathsf{ff}))))$ to $\mathrm{step}\,\mathrm{now}(\eta^T(\star))$. Therefore it must be the case that $\mathrm{op}(\mathrm{step}(\mathrm{now}(\eta^T(\mathsf{tt}))), \mathrm{step}(\mathrm{now}(\eta^T(\mathsf{ff}))))$ is $\mathrm{step}(\mathrm{now}(\mathrm{op}_1(\eta^T(\mathsf{tt}), \eta^T(\mathsf{ff}))))$ for some op_1. ◀

▶ **Proposition 29.** *There is no distributive law $\mathsf{P_f} D \to D\mathsf{P_f}$ for $\mathsf{P_f}$ the finite powerset functor.*

Proof Sketch. There are only four possible cases for $\mathrm{op}_1(x, y)$ and op'(x, y): $\emptyset, \{x\}, \{y\}$ and $\{x, y\}$. An easy analysis rules out the first three. Lemma 28 then implies that $\{\mathrm{step}(x), y\} = \mathrm{step}(\{x, y\})$. This leads to a contradiction as follows

$$\mathrm{step}(\{x\}) = \{\mathrm{step}(x)\} = \{\mathrm{step}(x), \mathrm{step}(x)\} = \mathrm{step}(\{x, \mathrm{step}(x)\}) = \mathrm{step}^2(\{x\}).$$ ◀

▶ **Example 30.** Let A be the algebraic theory with one idempotent binary operation $*$ and one unary operator !, with no further equations. Let T be the monad generated by A. There is a distributive law $\zeta : TD \to DT$ given by the following clauses

$$!(\mathrm{step}(x)) = x \qquad \mathrm{step}(x) * y = \mathrm{step}(x * (!y)) \qquad x * \mathrm{step}(y) = \mathrm{step}((!x) * y).$$

Note in particular, that $\mathrm{step}(x) * \mathrm{step}(y) = \mathrm{step}(x * (!(\mathrm{step}(y)))) = \mathrm{step}(x * y)$. This example can be extended to $*$ associative, if the equation $!(x * y) = (!x) * (!y)$ is added.

▶ **Theorem 31** (No-Go Theorem). *Let T be a monad with a binary algebraic operation that is commutative and idempotent. Then there is no distributive law of type $TD^\kappa \to D^\kappa T$.*

7 Semi-Go Theorem: Up to Weak Bisimilarity

In the proof of Theorem 31 the failure of existence of distributive laws comes down to a miscounting of steps. This section shows that this is indeed all that fails, and that parallel lifting defines a distributive law *up to weak bisimilarity* for algebraic monads with no drop equations. Weak bisimilarity is a relation on the coinductive delay monad, which relates computations that only differ by a finite number of steps. To make this precise, we work in a category of setoids. The objects are pairs (X, R), where R is an equivalence relation on X, and morphisms are equivalence classes of maps f between the underlying sets respecting the relations. Two such maps are equivalent if their values on equal input are related by the equivalence relation on the target type.

We first define a lifting of the coinductive delay monad D to the category of setoids. We do this via a similar relation (taken from Møgelberg and Paviotti [31]) defined for the guarded delay monad, because that allows us to reason using guarded recursion. We write δ^κ for $\mathrm{step} \circ \mathrm{next} : D^\kappa X \to D^\kappa X$.

▶ **Definition 32** ([31]). *Let X, Y be sets, and suppose $R : X \to Y \to \mathsf{Prop}$ is a relation. Define weak bisimilarity up to R, written $\sim^\kappa_R : D^\kappa X \to D^\kappa Y \to \mathsf{Prop}$, by:*

$$\begin{aligned}
\mathrm{now}(x) \sim^\kappa_R y &\stackrel{\mathit{def}}{=} \exists (n : \mathbb{N}, y' : Y). y = (\delta^\kappa)^n(\mathrm{now}(y'))\, \mathit{and}\ R(x, y'), \\
x \sim^\kappa_R \mathrm{now}(y) &\stackrel{\mathit{def}}{=} \exists (n : \mathbb{N}, x' : X). x = (\delta^\kappa)^n(\mathrm{now}(x'))\, \mathit{and}\ R(x', y), \\
\mathrm{step}(x) \sim^\kappa_R \mathrm{step}(y) &\stackrel{\mathit{def}}{=} \triangleright (\alpha\!:\!\kappa).(x\,[\alpha] \sim^\kappa_R y\,[\alpha]).
\end{aligned}$$

Note that the two first cases both apply for $\mathrm{now}(x) \sim^\kappa_R \mathrm{now}(y)$, but that they are equivalent in that case. If R is symmetric and reflexive, then the same properties hold for $\sim^\kappa_R$, but transitivity is not preserved. In fact, if $\sim^\kappa_=$ were transitive, then one could prove that it is the total relation, which is not the case.

▶ **Definition 33.** *Let* $R : X \to Y \to \mathsf{Prop}$ *be a relation. Define* $\sim_R : DX \to DY \to \mathsf{Prop}$ *as*

$$x \sim_R y \stackrel{def}{=} \forall \kappa. x\,[\kappa] \sim^{\kappa}_{R} y\,[\kappa].$$

The above definition is an encoding (using guarded recursion) of the standard coinductive definition of weak bisimilarity. We note the following, which was also observed by Chapman et al [10].

▶ **Proposition 34.** *The mapping* $D_{\mathsf{sd}}(X, R) = (DX, \sim_R)$ *defines a monad on the category of setoids.*

In fact, the multiplication for D^κ preserves the guarded recursive definition of weak bisimilarity.

Similarly, any algebraic monad T can be lifted to the category of setoids by defining $T(R)$ to be the smallest equivalence relation relating an equivalence class $[t(x_1, \ldots, x_n)]$ to $[t(y_1, \ldots, y_n)]$ if $R(x_i, y_i)$ for all i. We write T_{sd} for this.

Recall that by Proposition 22, if T is an algebraic monad given by a theory with no drop equations, then parallel lifting defines a natural transformation $TD \to DT$ on the category of sets. We show that this map lifts to a distributive law on the category of setoids.

▶ **Theorem 35.** *Let* T *be the free model monad of algebraic theory* $\mathbb{T} = (\Sigma_{\mathbb{T}}, E_{\mathbb{T}})$*, such that* $E_{\mathbb{T}}$ *contains no drop equations. Then parallel lifting defines a distributive law of monads* $T_{\mathsf{sd}} D_{\mathsf{sd}} \to D_{\mathsf{sd}} T_{\mathsf{sd}}$*.*

▶ Remark 36. The free monad on an algebraic theory could alternatively be expressed on the category of setoids by taking the set to be the free monad just on operations, introducing the equations of the theory into the equivalence relation. In the presence of the axiom of choice this generates a monad equivalent to T_{sd}, and we expect that the proof above can be adapted to that choice as well.

8 Related Work

Møgelberg and Vezzosi [32] study two combinations of the guarded delay monad D^κ with the finite powerset monad $\mathsf{P_f}$ expressed as a HIT in CCTT. They use these to show that applicative simulation is a congruence for the untyped lambda calculus with finite non-determinism using denotational techniques. One combination is the sum $\mathsf{P_f} \oplus D^\kappa$, which is used for the case of may-convergence, and the other is the composite $D^\kappa \mathsf{P_f}$ equipped with the parallel lifting, which is used for must-convergence. They observe that only the former is a monad. In this paper, we not only provide a more general study of such combinations, but also suggest a way to remedy the situation in the latter case by considering weak bisimilarity.

Weak bisimilarity for the coinductive delay monad was first defined by Capretta [8]. Møgelberg and Paviotti [31] show that their embedding of FPC in guarded dependent type theory respects weak bisimilarity and use that to prove an adequacy theorem up to weak bisimilarity.

Chapman et al. [10] observe that quotienting the coinductive delay monad by weak bisimilarity appears to not yield a monad unless countable choice is assumed. Altenkirch et al. [1] propose a solution to this problem by constructing the quotient and the weak bisimilarity relation simultaneously, as a higher inductive-inductive type. Chapman et al. themselves suggest a different solution, constructing the quotient as the free ω-cpo using an ordinary HIT. These quotients have not (to the best of our knowledge) been studied in combination with other effects.

Interaction trees [42] are essentially monads of the form $\forall\kappa.(T \oplus D^\kappa)(-)$ for T an algebraic monad generated by operations with no equations. Much work has gone into building libraries for working with these up to weak bisimilarity in Coq, and these allow for interaction trees to be used for program verification. To our knowledge, versions of interaction trees with equations between terms have not been considered.

As mentioned in the introduction, the guarded recursive delay monad has two benefits over the coinductive one: Firstly, it has a fixed point operator of the type (rather than an iteration operator), which means that it allows for embedding languages with recursion directly in type theory. In the coinductive case, one must either use some encoding of recursion using the iteration operator, or prove that all constructions used are continuous. We believe this is a considerable burden for higher order functions. The second advantage is that guarded recursion allows for also advanced notions of state to be encoded, as shown recently by Sterling et al. [38]. Neither the interaction trees nor the quotiented delay monads appear to have these benefits.

Related Work on Monad Compositions

The field of monad compositions in general has attracted quite a bit of attention lately. After Plotkin proved that there is no distributive law combining probability and non-determinism [40], Klin and Salamanca [21] studied impossible distributions of the powerset monad over itself, while Zwart and Marsden provided a general study on what makes distributive laws fail [43]. Meanwhile, the initial study of monad compositions by Manes and Mulry [26, 27] was continued by Parlant et al [13, 34, 35]. In both the positive and the negative theorems on distributive laws in these papers, certain classes of equations were identified as causes for making or breaking the monad composition. Idempotence, duplication, and dropping variables came out as especially noteworthy types of equations, which the findings in this paper confirm.

Our study of the delay monad provides an interesting extension on the previous works, because of its non-standard algebraic structure given by delay algebras, and the fact that the delay monad is neither affine nor relevant, which are the main properties studied by Parlant et al.

9 Conclusion and Future Work

We have studied how both the guarded recursive and the coinductive version of the delay monad combine with other monads. After studying some specific examples and free combinations, we looked more generally at possible distributive laws of $TD^\kappa \to D^\kappa T$. We found two natural candidates for such distributive laws, induced by *parallel* and *sequential* lifting of operations on T. We showed that:

- Sequential lifting provides a distributive law for monads presented by theories with balanced equations.
- There is no distributive law possible for monads with a binary operation that is commutative and idempotent over D^κ, but this does not rule out a distributive law of such monads over D.
- Parallel lifting does not define a distributive law, but it does define one *up to weak bisimilarity*, for monads presented by theories with non-drop equations.

It is unfortunate that weak bisimilarity requires working with setoids, but this is due to the quotient of D up to weak bisimilarity not being a monad [10]. It is not clear how to adapt the solutions to this problem mentioned above [10, 1] to the guarded recursive setting.

This paper only considers the case of finite arity operations (except for state, which can be of any arity). Distributive laws for countable arity operations such as countable non-deterministic choice are more difficult. In those cases sequential lifting seems an unnatural choice, not only because it does not interact well with idempotency, but also because it introduces divergence even in the cases where there is an upper limit to the number of steps taken by the arguments. Extending our parallel lifting operation to the countable case requires deciding whether all the countably many input operations are values, which is not possible in type theory.

The results presented in this paper are formulated and proven in CCTT. It is natural to ask whether the results proven for the coinductive delay monad D also hold for D considered as a monad on the category Set of sets. For some of the results proven in this paper (Proposition 29 and Example 30) both the statements and proofs can be read in Set. These results can therefore easily be seen to hold in this setting. In many other cases, our constructions use guarded recursion (e.g. the definitions of parallel and sequential lifting of operators). To lift these results to Set, one would need to redo the constructions and argue for their productivity. However, we believe that using guarded recursion is the natural way to work with coinductive types and proofs. Another approach could therefore be to use guarded recursion as a language to reason about Set. This should be possible because the universe used to model clock irrelevant types in the extensional model of Clocked Type Theory [28] classifies a category equivalent to Set. We leave this as a direction for future research.

References

1. Thorsten Altenkirch, Nils Anders Danielsson, and Nicolai Kraus. Partiality, revisited. In *International Conference on Foundations of Software Science and Computation Structures*, pages 534–549. Springer, 2017.
2. Robert Atkey and Conor McBride. Productive coprogramming with guarded recursion. *ACM SIGPLAN Notices*, 48(9):197–208, 2013.
3. Patrick Bahr., Hans Bugge Grathwohl, and Rasmus Ejlers Møgelberg. The clocks are ticking: No more delays! In *2017 32nd Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*, pages 1–12. IEEE, 2017.
4. Patrick Bahr and Graham Hutton. Monadic compiler calculation (functional pearl). *Proc. ACM Program. Lang.*, 6(ICFP), August 2022. doi:10.1145/3547624.
5. Magnus Baunsgaard Kristensen, Rasmus Ejlers Mogelberg, and Andrea Vezzosi. Greatest hits: Higher inductive types in coinductive definitions via induction under clocks. In *Proceedings of the 37th Annual ACM/IEEE Symposium on Logic in Computer Science*, pages 1–13, 2022.
6. Jon Beck. Distributive laws. In B. Eckmann, editor, *Seminar on Triples and Categorical Homology Theory*, pages 119–140. Springer Berlin Heidelberg, 1969.
7. Lars Birkedal, Rasmus Ejlers Møgelberg, Jan Schwinghammer, and Kristian Støvring. First steps in synthetic guarded domain theory: step-indexing in the topos of trees. *Logical Methods in Computer Science*, 8(4), 2012.
8. Venanzio Capretta. General recursion via coinductive types. *Logical Methods in Computer Science*, 1, 2005.
9. Evan Cavallo and Robert Harper. Higher inductive types in cubical computational type theory. *Proceedings of the ACM on Programming Languages*, 3(POPL):1–27, 2019.
10. James Chapman, Tarmo Uustalu, and Niccolò Veltri. Quotienting the delay monad by weak bisimilarity. *Mathematical Structures in Computer Science*, 29(1):67–92, 2019.
11. Ranald Clouston. Fitch-style modal lambda calculi. In *International Conference on Foundations of Software Science and Computation Structures*, pages 258–275. Springer, 2018.

12 Cyril Cohen, Thierry Coquand, Simon Huber, and Anders Mörtberg. Cubical type theory: A constructive interpretation of the univalence axiom. In *21st International Conference on Types for Proofs and Programs (TYPES 2015)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2018.

13 Fredrik Dahlqvist, Louis Parlant, and Alexandra Silva. Layer by layer - combining monads. In *Theoretical Aspects of Computing – ICTAC 2018 – 15th International Colloquium, Stellenbosch, South Africa, October 16-19, 2018, Proceedings*, pages 153–172, 2018. doi:10.1007/978-3-030-02508-3_9.

14 Nils Anders Danielsson. Operational semantics using the partiality monad. *SIGPLAN Not.*, 47(9):127–138, September 2012. doi:10.1145/2398856.2364546.

15 Martín Escardó and Paulo Oliva. Selection functions, bar recursion and backward induction. *Mathematical Structures in Computer Science*, 20(2):127–168, 2010. doi:10.1017/S0960129509990351.

16 ND Gautam. The validity of equations of complex algebras. *Archiv für mathematische Logik und Grundlagenforschung*, 3(3):117–124, 1957.

17 Daniel Gratzer, GA Kavvos, Andreas Nuyts, and Lars Birkedal. Multimodal dependent type theory. In *Proceedings of the 35th Annual ACM/IEEE Symposium on Logic in Computer Science*, pages 492–506, 2020.

18 Jules Hedges. Monad transformers for backtracking search. In Paul Levy and Neel Krishnaswami, editors, Proceedings 5th Workshop on *Mathematically Structured Functional Programming,* Grenoble, France, 12 April 2014, volume 153 of *Electronic Proceedings in Theoretical Computer Science*, pages 31–50. Open Publishing Association, 2014. doi:10.4204/EPTCS.153.3.

19 Martin Hyland, Gordon Plotkin, and John Power. Combining effects: Sum and tensor. *Theoretical Computer Science*, 357(1):70–99, 2006. doi:10.1016/j.tcs.2006.03.013.

20 Martin Hyland and John Power. Discrete lawvere theories and computational effects. *Theoretical Computer Science*, 366(1-2):144–162, 2006.

21 Bartek Klin and Julian Salamanca. Iterated covariant powerset is not a monad. In *Proceedings 34th Conference on the Mathematical Foundations of Programming Semantics, MFPS 2018*, 2018.

22 Anders Kock. Monads on symmetric monoidal closed categories. *Archiv der Mathematik*, 21(1):1–10, 1970.

23 F. William Lawvere. Functorial semantics of algebraic theories. *Proceedings of the National Academy of Sciences of the United States of America*, 50(5):869–872, 1963.

24 Fred EJ Linton. Some aspects of equational categories. In *Proceedings of the Conference on Categorical Algebra*, pages 84–94. Springer, 1966.

25 Ernie Manes. *Algebraic theories*, volume 26. Springer, 1976.

26 Ernie Manes and Philip Mulry. Monad compositions I: general constructions and recursive distributive laws. *Theory and Applications of Categories*, 18:172–208, April 2007.

27 Ernie Manes and Philip Mulry. Monad compositions II: Kleisli strength. *Mathematical Structures in Computer Science*, 18(3):613–643, 2008. doi:10.1017/S0960129508006695.

28 Bassel Mannaa, Rasmus Ejlers Møgelberg, and Niccolò Veltri. Ticking clocks as dependent right adjoints: Denotational semantics for clocked type theory. *Logical Methods in Computer Science*, 16, 2020.

29 Per Martin-Löf. *Intuitionistic Type Theory*. Bibliopolis, Napoli, 1984.

30 Lambert Meertens. Algorithmics, towards programming as a mathematical activity. In J.W. De Bakker, M. Hazewinkel, and J.K. Lenstra, editors, *Mathematics and Computer Science: Proceedings of the CWI Symposium, November 1983*, CWI monographs, pages 289–334. North-Holland, 1986.

31 Rasmus E Møgelberg and Marco Paviotti. Denotational semantics of recursive types in synthetic guarded domain theory. *Mathematical Structures in Computer Science*, 29(3):465–510, 2019.

32 Rasmus Ejlers Møgelberg and Andrea Vezzosi. Two guarded recursive powerdomains for applicative simulation. In Ana Sokolova, editor, *Proceedings 37th Conference on Mathematical Foundations of Programming Semantics, MFPS 2021, Hybrid: Salzburg, Austria and Online, 30th August - 2nd September, 2021*, volume 351 of *EPTCS*, pages 200–217, 2021. `doi:10.4204/EPTCS.351.13`.

33 Hiroshi Nakano. A modality for recursion. In *Proceedings Fifteenth Annual IEEE Symposium on Logic in Computer Science*, pages 255–266. IEEE, 2000.

34 Louis Parlant. *Monad Composition via Preservation of Algebras*. PhD thesis, UCL (University College London), 2020.

35 Louis Parlant, Jurriaan Rot, Alexandra Silva, and Bas Westerbaan. Preservation of equations by monoidal monads. In *45th International Symposium on Mathematical Foundations of Computer Science (MFCS 2020)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2020.

36 Marco Paviotti, Rasmus Ejlers Møgelberg, and Lars Birkedal. A model of PCF in guarded type theory. *Electronic Notes in Theoretical Computer Science*, 319:333–349, 2015.

37 Gordon Plotkin and John Power. Notions of computation determine monads. In *International Conference on Foundations of Software Science and Computation Structures*, pages 342–356. Springer, 2002.

38 Jonathan Sterling, Daniel Gratzer, and Lars Birkedal. Denotational semantics of general store and polymorphism. Unpublished manuscript, July 2022.

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

40 Daniele Varacca and Glynn Winskel. Distributing probability over non-determinism. *Mathematical Structures in Computer Science*, 16(1):87–113, 2006.

41 Niccolò Veltri and Niels F. W. Voorneveld. Inductive and coinductive predicate liftings for effectful programs. In Ana Sokolova, editor, *Proceedings 37th Conference on Mathematical Foundations of Programming Semantics, MFPS 2021, Hybrid: Salzburg, Austria and Online, 30th August - 2nd September, 2021*, volume 351 of *EPTCS*, pages 260–277, 2021. `doi:10.4204/EPTCS.351.16`.

42 Li-yao Xia, Yannick Zakowski, Paul He, Chung-Kil Hur, Gregory Malecha, Benjamin C. Pierce, and Steve Zdancewic. Interaction trees: Representing recursive and impure programs in Coq. *Proc. ACM Program. Lang.*, 4(POPL), December 2019. `doi:10.1145/3371119`.

43 Maaike Zwart and Dan Marsden. No-go theorems for distributive laws. In *34th Annual ACM/IEEE Symposium on Logic in Computer Science (LiCS)*, pages 1–13, 2019. `doi:10.1109/LICS.2019.8785707`.