Theoretical Informatics and Applications Will be set by the publisher
Informatique Théorique et Applications

COPRODUCTS OF IDEAL MONADS

NEIL GHANI[1] AND TARMO UUSTALU[2]

Abstract. The question of how to combine monads arises naturally in many areas with much recent interest focusing on the coproduct of two monads. In general, the coproduct of arbitrary monads does not always exist. Although a rather general construction was given by Kelly [15], its generality is reflected in its complexity which limits the applicability of this construction. Following our own research [19], and that of Hyland, Plotkin and Power [12], we are looking for specific situations when simpler constructions are available. This paper uses fixed points to give a simple construction of the coproduct of two *ideal monads*.

1991 Mathematics Subject Classification. 08B20,18C15,18C50,68Q55.

1. Introduction

Monads are a fundamental concept in category theory. One of their first applications was to give a categorical account of universal algebra [17] and they have since remained central to the development of category theory. More recently they have found further significant applications in computer science as they provide i) an abstract model of syntax covering algebraic theories [21], higher-order abstract syntax [7] and non-wellfounded syntax [1,8,26] etc. where the term algebra construction, substitution and variables are taken as primitive, ii) a useful abstraction of computational effects permitting a uniform treatment of diverse features of impure programming languages such as stateful computations, exceptions and I/O [24], and iii) a semantic framework for modularity where the interaction of different components of a complex system is modelled by the interleaving of the representing monads [20].

In each of the three applications above, we want to combine monads. In the case of monads as abstract models of syntax we are often interested in combining

[1] Dept. of Math. and Comp. Sci., University of Leicester, University Road, Leicester LE1 7RH, UK, e-mail: `ng13@mcs.le.ac.uk`.
[2] Inst. of Cybernetics, Tallinn Technical University, Akadeemia tee 21, EE-12618 Tallinn, Estonia, e-mail: `tarmo@cs.ioc.ee`.

different theories as is often practiced in the area of algebraic specification [11]. In functional programming, how can we combine the state monad and the exceptions monad to model a program which has both stateful computations and which may raise exceptions [14]? Finally, in the semantics of modularity, if two systems (e.g., term rewriting systems) R and S are modelled by monads T_R and T_S, how can we reason about the combined system $R + S$ via its representing monad T_{R+S}? That is, how can we express T_{R+S} in terms of T_R and T_S [20]? A variety of different methods to combine monads have been proposed, most notably in the use of *distributive laws* [5,6] and *monad transformers* [14]. While being useful in specific situations, these theories do not cover all situations, and furthermore, as we comment later, can sometimes seem rather ad hoc.

These approaches observe that monads are functors carrying extra structure. Thus the obvious way to compose a monad R and a monad S is to use the functorial composition SR. However, SR is not a monad in general and one can see both distributive laws and monad transformers as an attempt to coax SR into a monad. An alternative point of view is that, just as functors are the objects of the functor category, so monads are the objects of the category of monads and monad morphisms. Then the canonical way of putting two objects together is take their *coproduct*. While it is not the case that the coproduct of arbitrary monads always exists, they do so under mild conditions as given in the rather general construction of coproducts (even colimits) of monads given by Kelly [15]. However, the generality of the construction is reflected in its complexity which can be deterring even for experienced category theorists and which certainly limits its applicability. Consequently, recent research has focussed on providing alternative constructions which, by restricting to special cases, are significantly simpler and hence easier to apply in practice. In particular, we would like to reduce the coproduct of monads to formulae involving fixed point constructions and functorial operations such as composition and coproduct. As we remark in Section 2, this can be done for the coproduct of free monads and, as proven by Hyland, Plotkin and Power [12], the coproduct of a free monad with any other monad. We make further comments relating our work to that of Hyland, Plotkin and Power in Section 5 concerning further work.

Most monads, including many important examples, are not free and this limits the applicability of the results above. In particular, this paper arose out of research into modularity in i) non-wellfounded syntax and ii) higher-order syntax. In the former we want to study coproducts of free completely iterative monads as these model non-wellfounded terms [2, 26] while in the latter we study monads which model calculi with variable binding such as the λ-calculus [7] and which typically arise as least fixed points of higher-order functors. The results above do not apply to coproducts of such monads since these monads are not free. To understand the coproduct of such monads, we return to the central intuition of *layers* that we used in our semantic proofs [20] of modularity in term rewriting. The coproduct construction attempts to interleave these layers but, in general, layers can collapse, which causes mathematical difficulties necessitating the complexities of Kelly's constructions. However, in most examples of interest, including those above, layers do not collapse. Intriguingly, the *ideal monads*, introduced by Adámek et al. [1] to study guarded recursion, provide exactly the right mathematical structure to capture this idea of layers not collapsing and hence our abstract result is the reduction of the coproduct of ideal monads to fixed point formulae. Accordingly, the general thrust in this paper is to avoid commitment to a specific semantic structure, e.g., finitary monads over lfp categories, but rather to use the existence of various initial algebras as the preconditions of our constructions.

To summarise, the field of modularity in both syntax and computational phenomena is highly promising but has been held back by the lack of simple mechanisms for computing coproducts of naturally arising monads. This paper provides just the right kind of result to spur research in this area. The paper is structured as follows. In Section 2, we recall the basic results about monads which we use in the rest of the paper. Section 3 then introduces ideal monads and contains our main results while Section 4 applies them to the question of modularity of non-wellfounded syntax and higher-order syntax. We finish in Section 5 with some concluding remarks and directions for future research.

2. Monads, Coproducts and Layers

Monads originally arose within category theory as models of term algebras and algebraic theories. Such monads provide good intuitions as to how to construct the coproduct of two monads and so we begin with them. Since this is standard material, we refer the reader to general texts [5, 21] for more details.

2.1. Monads and Term Algebras

Term algebras are built from signatures which are defined as follows:

Definition 2.1. A (single-sorted) *signature* consists of a function $\Sigma : \mathbb{N} \to \mathbf{Set}$. The set of *$n$-ary operators* of Σ is defined $\Sigma_n = \Sigma(n)$.

Definition 2.2. Given a signature Σ and a set of variables X, the *term algebra* $T_\Sigma(X)$ is the set defined inductively by the following rules:

$$\frac{x \in X}{{}'x \in T_\Sigma(X)} \qquad\qquad \frac{f \in \Sigma_n \quad t_1, \ldots, t_n \in T_\Sigma(X)}{f(t_1, \ldots, t_n) \in T_\Sigma(X)}$$

We use quotes to distinguish a variable $x \in X$ from the term ${}'x \in T_\Sigma(X)$ and this can be seen as introducing layer information into terms. As we shall see later, when constructing the coproduct of two monads, this layer structure is the central concept.

Categorically, T_Σ is an endofunctor on $\mathbf{Set}$ that can be constructed in one of two ways. Start by defining the associated polynomial endofunctor $F_\Sigma : \mathbf{Set} \to \mathbf{Set}$ by $F_\Sigma(X) = \coprod_{n \in \mathbb{N}, f \in \Sigma_n} X^n$. Then:

- either, for each set X, $T_\Sigma X$ is defined to be the carrier $\mu(X + F_\Sigma(\_))$ of the initial algebra (least fixed point) of the endofunctor $X + F_\Sigma(\_)$ on $\mathbf{Set}$,

- or, T_Σ is equivalently defined in a non-pointwise fashion as the carrier $\mu(\mathsf{Id} + F_\Sigma \circ \_)$ of the initial algebra of the endofunctor $\mathsf{Id} + F_\Sigma \circ \_$ on $[\mathbf{Set}, \mathbf{Set}]$.

Either way, for every set of variables X, there is a function $X \to T_\Sigma(X)$ sending each variable x to the associated term *'x*. Lastly, substitution takes terms built over terms and flattens them, as described by a function $T_\Sigma T_\Sigma(X) \to T_\Sigma(X)$. Both two operations are natural in X. These three pieces of data, namely the construction of a term algebra from a set of variables, the embedding of variables as terms and the operation of substitution are axiomatised in the concept of a monad.

Definition 2.3 (Monads)**.** A *monad* on a category $\mathcal{C}$ is an endofunctor $T : \mathcal{C} \to \mathcal{C}$ together with two natural transformations, $\eta : \mathsf{Id} \to T$, called the *unit*, and $m : TT \to T$, called the *multiplication* of the monad, such that the following diagrams, known as the post-unit, pre-unit and associativity of multiplication laws, commute.

$$\begin{tikzcd} T \arrow[r, "\eta T"] \arrow[dr, equal] & TT \arrow[d, "m"] \\ & T \end{tikzcd} \qquad \begin{tikzcd} T \arrow[d, "T\eta"'] \arrow[dr, equal] & \\ TT \arrow[r, "m"'] & T \end{tikzcd} \qquad \begin{tikzcd} TTT \arrow[r, "mT"] \arrow[d, "Tm"'] & TT \arrow[d, "m"] \\ TT \arrow[r, "m"'] & T \end{tikzcd}$$

□

We write m for the multiplication rather than the usual μ, since we reserve μ for least fixed points. Although a monad is fully specified only by all of the data (T, η, m), it is customary to refer it to by simply T, when it is clear that the monad and not just the functor is meant and the data η, m are clear from the context—a convention which we follow here. As suggested above, the term algebra functor T_Σ underlies a monad for any signature Σ. In fact, we can even make a slightly more general statement, abstracting from **Set** to any category $\mathcal{C}$ and from a polynomial endofunctor F_Σ on **Set** determined by a signature Σ to any endofunctor F on $\mathcal{C}$.

Proposition 2.4. *Let $\mathcal{C}$ have coproducts. If the free algebra functor $T_F = \mu(\mathsf{Id} + F \circ \_)$ of an endofunctor F on a category $\mathcal{C}$ is defined (the initial algebra exists), then T_F is the underlying functor of a monad.*

Lemma 2.4 is typical of the reasoning in this paper in that it is based on properties of initial algebras. Thus, in general, we shall assume the existence of such fixed points whenever we speak of them and thereby avoid commitment to a specific semantic structure which guarantees the existence of these fixed points. Of course, in many cases, restricting to ω-cocontinuous endofunctors over categories with ω-colimits suffices to guarantee the existence of the fixed points we require.

Monads also model a number of other important structures in computer science, such as (many-sorted) algebraic theories, non-wellfounded syntax [1, 8, 26], term graphs [9], calculi with variable binders [7], term rewriting systems [18], and, via computational monads [24], state-based computations, exceptions, continuations etc. These applications involve base categories other than **Set** and the desire for a uniform treatment underpins their monadic axiomatisation.

In order to talk about coproducts of monads, we must of course define the category of monads.

Definition 2.5 (Monad morphisms)**.** Given two monads (T, η, m) and (H, η^H, m^H) on $\mathcal{C}$ a *monad morphism* is a natural transformation $h : T \to H$ preserving the unit and multiplication in the sense of commutation of the diagrams

$$\begin{tikzcd} \mathsf{Id} \arrow[r, "\eta"] \arrow[dr, "\eta^H"'] & T \arrow[d, "h"] \\ & H \end{tikzcd} \qquad \begin{tikzcd} TT \arrow[r, "m"] \arrow[d, "hh"'] & T \arrow[d, "h"] \\ HH \arrow[r, "m^H"'] & H \end{tikzcd}$$

□

The monads on a category $\mathcal{C}$ and monad morphisms between them form a category $\mathbf{Mon}(\mathcal{C})$.

The observation that T_Σ is a monad for a signature Σ does not capture the inductive nature of the term algebra construction. We now proceed to addressing this point by giving a proper characterization of the term algebra construction as a *free* monad. In general, free monads do not exist over all endofunctors (e.g., the powerset functor on **Set**) and so we define the free monad over a functor via *universal arrows.*

Definition 2.6 (Free monads)**.** A *free monad* over an endofunctor F on $\mathcal{C}$ is an universal arrow from F to the forgetful functor $U : \mathbf{Mon}(\mathcal{C}) \to [\mathcal{C}, \mathcal{C}]$, i.e., a monad (T, η, m) together with a natural transformation $\iota : F \to T$ such that for any monad (H, η^H, m^H) on $\mathcal{C}$ and a natural transformation $f : F \to H$, there is a unique monad morphism $h : (T, \eta, m) \to (H, \eta^H, m^H)$ such that the diagram

$$\begin{tikzcd} F \arrow[r, "\iota"] \arrow[dr, "f"'] & T \arrow[d, "h"] \\ & H \end{tikzcd}$$

commutes. □

The following observation is standard knowledge since [4].

Proposition 2.7. *If, for an endofunctor F on a category $\mathcal{C}$, the functor $T_F = \mu(\mathsf{Id} + F \circ \_)$ is defined (the initial algebra exists), then the monad structure on T_F is the free monad over F.*

The classic special case is that of term algebras for a signature. The Σ-term algebra functor T_Σ is not just a monad, it is the free monad: It is not just some Σ-algebra functor containing the variables (the identity functor) and closed under substitution, it is the least such.

2.2. Coproducts of monads

We now turn to coproducts of monads. By the coproduct of two monads R, S, we mean of course the coproduct of R and S as objects in the category of monads, which, intuitively, must be the least monad containing both in a disjoint manner. We write $\oplus$ for the coproduct of monads and reserve $+$ for the functorial coproduct. The first result on coproducts of monads concerns free monads.

Proposition 2.8 (Coproduct of free monads)**.** *Let T_F and T_G be free monads. Then, if the free monad T_{F+G} exists, it is the coproduct of T_F and T_G.*

Proof. The universal property of the coproduct follows directly from the universal properties of T_F and T_G. $\square$

Hyland, Plotkin and Power [12] have given the following construction for the coproduct of a free monad with an arbitrary monad.

Proposition 2.9 (Coproduct of a free monad with a monad)**.** *Let T_F be a free monad and S a monad. If the free monad T_{FS} exists, then ST_{FS} is the underlying functor of the coproduct of T_F and S.*

The above functorial expression can be rewritten into an equivalent form where S appears only once: we have $ST_{FS} = S\mu(\mathsf{Id} + FS \circ \_)$ by the least fixed point construction of the free monad and $S\mu(\mathsf{Id} + FS \circ \_) \cong \mu(S(\mathsf{Id} + F \circ \_))$ by the folklore rolling lemma. Again, all that is needed for these equations to hold is the existence of the various initial algebras.

The coproduct of free monads over signatures gives good intuitions as to how to construct coproducts of arbitrary monads. Given two signatures Σ and Ω with corresponding term algebra monads T_Σ and T_Ω, we have seen that $T_\Sigma \oplus T_\Omega = T_{\Sigma+\Omega}$, that is the coproduct $T_\Sigma \oplus T_\Omega$ should calculate the terms built over the disjoint union of the signatures. Terms in $T_{\Sigma+\Omega}(X)$ have an inherent notion of layer, as a term in $T_{\Sigma+\Omega}(X)$ either is a variable or decomposes into a term from T_Σ (or T_Ω), and strictly smaller subterms with head symbols from Ω (or Σ respectively). This suggests that we can build the underlying functor of the coproduct $T_{\Sigma+\Omega}(X)$ by successively applying the underlying functors T_Σ and T_Ω (but see also the discussion below, there is a correction to be made to the formula we give here):

$$
\begin{array}{rcl}
(T_\Sigma \oplus T_\Omega)(X) & = & X \\
& + & T_\Sigma(X) + T_\Omega(X) \\
& + & T_\Sigma T_\Omega(X) + T_\Omega T_\Sigma(X) \\
& + & T_\Sigma T_\Omega T_\Sigma(X) + \ldots
\end{array}
$$

In the above summation, an element of $T_\Sigma T_\Omega T_\Sigma(X)$ can be thought of as having a top T_Σ-layer followed by a T_Ω-layer followed by another T_Σ-layer. Indeed, from now on, we abstract from the set of variables and work at the functorial/monadic level so as to talk about these layers. In general, for any monad T, we often regard $T(X)$ abstractly as a layer; in the examples above, layers were terms, rewrites, or computations. Monads can then be seen as abstracting from the nature of a layer and thereby provide a calculus for manipulating layers where the types of layers, the trivial layer of each type, and the collapsing of two layers of the same type are are taken as primitive concepts. As shown in the equation above, the coproduct intuitively consists of all finite interleavings of layers from the summands.

Now, note that there is a natural transformation

$$T_\Sigma \eta^\Omega T_\Sigma : T_\Sigma T_\Sigma \to T_\Sigma T_\Omega T_\Sigma$$

which essentially puts a trivial layer between two adjacent T_Σ-layers. Of course terms in the image of $T_\Sigma \eta^\Omega T_\Sigma$ really should be counted as only having one T_Σ-layer, since they contain no Ω-constructors. Thus, in the right-hand side of our coproduct formula above we should really have used $F_\Sigma T_\Sigma$ and $F_\Omega T_\Omega$ instead of T_Σ, T_Ω. And, more generally, we really want to be able to talk about non-trivial layers, or that part of a monad which is not in the image of the unit. This is exactly our motivation to focus on *ideal monads*, which we will do next.

3. Ideal Monads and Their Coproducts

This section is devoted to *ideal monads* [1]—a large variety of monads that covers many familiar types of monads capturing notions of syntax and notions of computation—and a fixed point construction for calculating the coproduct of two ideal monads. Although ideal monads were introduced for exactly the reason we use them, namely to separate the variable and non-variable parts, the application in mind was to guarded recursion. The fact that we have a separate use for them makes us think they may be applicable in many situations.

3.1. Ideal Monads

Ideal monads are monads in which the image of the unit is separated from the rest of the monad.

Definition 3.1 (Ideal monads)**.** An *ideal monad* on a category $\mathcal{C}$ is a monad (T, η, m) on $\mathcal{C}$ together with an endofunctor T_0 on $\mathcal{C}$ and a natural transformation $m_0 : T_0 T \to T_0$ such that T is the coproduct $\mathsf{Id} + T_0$, the unit η is the left injection $\mathsf{inl}_{\mathsf{Id}, T_0}$ and the square

$$\begin{tikzcd} T_0 T \arrow[r, "\mathsf{inr}_{\mathsf{Id},T_0} T"] \arrow[d, "m_0"'] & TT \arrow[d, "m"] \\ T_0 \arrow[r, "\mathsf{inr}_{\mathsf{Id},T_0}"'] & T \end{tikzcd}$$

commutes (which is equivalent to requiring that $m = [T, \mathsf{inr}_{\mathsf{Id},T_0} \cdot m_0]$). $\square$

We can see that an ideal monad (T, η, m, T_0, m_0) is fully specified by its data (T_0, m_0) and that any endofunctor T_0 together with a natural transformation m_0 :

$T_0T \to T_0$ yields an ideal monad provided that $T_0 = m_0 \cdot T_0\eta$ and $m_0 \cdot m_0T = m_0 \cdot T_0m$. These two conditions ensure η being a pre-unit of m and m being associative; that η is a post-unit of m is automatic. Just as we frequently refer to a monad (T, η, m) by simply T, we agree to refer to the ideal monad given by (T_0, m_0) by $\mathsf{Id} + T_0$ and leave the restricted form of multiplication m_0 implicit whenever possible.

A monad morphism $h : T \to H$ whose source is an ideal monad $T = \mathsf{Id} + T_0$ has its action on Id forced by the preservation of the unit requirement and is hence of the form $[\eta^H, h_0]$ where $h_0 : T_0 \to H$. Given any $h_0 : T_0 \to H$, the natural transformation $h = [\eta^H, h_0]$ is a monad morphism iff $h_0 \cdot m_0^T = m^H \cdot h_0h$ (this condition guarantees that h preserves the multiplication).

Syntax-motivated examples of ideal monads include free monads, free completely iterative monads, monads arising as fixed points of higher order endofunctors etc.

Example 3.2 (Free monads)**.** Recall that the free monad T_F over an endofunctor F on a category $\mathcal{C}$ is given by $T_F = \mu(\mathsf{Id} + F \circ \_) \cong \mathsf{Id} + FT_F$, if this least fixed point exists, and that this covers the term algebra generated by a signature. It is straightforward to check that this monad is ideal—we already have the decomposition of the monad as a sum and it is straightforward to check the multiplication restricts appropriately. Informally, we know that every term is either a variable or a non-variable/operator application. $\square$

Example 3.3 (Free completely iterative monads)**.** An ideal monad $T = \mathsf{Id} + T_0$ on a category $\mathcal{C}$ is *completely iterative* iff, for any morphism $e : X \to Y + T_0(X + Y)$ in $\mathcal{C}$, there exists a unique morphism $e^\dagger : X \to T(Y)$ in $\mathcal{C}$ such that the diagram

$$\begin{tikzcd} Y + T_0(X+Y) \arrow[d, "Y + T_0[e^\dagger, \eta_Y]"'] & X \arrow[l, "e"'] \arrow[d, "e^\dagger"] \\ Y + T_0T(Y) \arrow[r, "{[\eta_Y, \mathsf{inr}_{\mathsf{Id}, T_0} \cdot m_{0_Y}]}"'] & T(Y) \end{tikzcd}$$

commutes. This concept was introduced by [1] to study the situation where every guarded system of recursive equations has a unique solution. The free (among the ideal monads) completely iterative monad T_F^∞ over an endofunctor F on $\mathcal{C}$ is $T_F^\infty = \nu(\mathsf{Id} + F \circ \_) \cong \mathsf{Id} + FT_F^\infty$, if this greatest fixed point exists (e.g., if $\mathcal{C}$ is ω-complete and F is ω-continuous). If $\mathcal{C} = \mathbf{Set}$ and $F = F_\Sigma$ for some first-order signature Σ, then T_F^∞ collects all non-wellfounded (i.e., possibly infinite) Σ-terms (leaf-labelled Σ-trees).

Switching from arbitrary guarded equation systems to those that are finitary (finitely many equations, all right-hand sides finite), one gets a wider subclass of ideal monads—*iterative monads*. The free iterative monad induced by a first-order signature Σ is the algebra of rational Σ-terms and is again ideal. Rational Σ-terms are those non-wellfounded Σ-terms which have a finite number of distinct subterms. $\square$

Example 3.4 (λ-calculus)**.** The language of untyped λ-calculus (à la de Bruijn, with free variables identified by their positions in the context, not their names) is described by the endofunctor $L = \mu(\mathsf{Id} + \Delta \circ \_ + \_ \circ \delta)$ on $\mathcal{C}$ where the higher-order functors Δ, $\delta : [\mathcal{C},\mathcal{C}] \to [\mathcal{C},\mathcal{C}]$ are given by $\Delta = \mathsf{Id} \times \mathsf{Id}$ and $\delta = K_1 + \mathsf{Id}$ (1 denoting the terminal object of $\mathcal{C}$, K_A denoting the constant functor returning A). L is certainly well defined and a monad in the critical test case of $\mathcal{C} = \mathbf{Set}$, while conditions to generalise these results to other categories are not very prohibitive. In the isomorphism $L \cong \mathsf{Id} + \Delta L + L\delta$, the second summand $\Delta L = L \times L$ corresponds to application terms and the third summand $L\delta = L(K_1 + \mathsf{Id})$ to λ-abstraction terms: for a raw λ-abstraction to be well formed wrt. a context, its body has to be well formed wrt. its extension with one extra variable that corresponds to the bound variable of the λ-abstraction. The monad L is not free, but it is ideal. $\square$

Example 3.5 (Explicit substitutions)**.** Much the same way as we defined the language of the λ-calculus as a least fixed point in a functor category, we can define the language of applications and explicit substitutions. Under identification of expressions equal up to a certain notion of α-equivalence for explicit substitutions, this is the endofunctor $E = \mu(\mathsf{Id} + \Delta \circ \_ + \_ \circ \_)$ on $\mathcal{C}$. This is apparent from the fact that $EE(Y) \cong (\mathsf{Lan}_{\mathsf{Id}} E)E(Y) \cong \int^X \mathcal{C}(X, E(Y)) \otimes E(X)$, i.e., we have a term in bound variables X and a map sending the bound variables to other terms. Under mild conditions, the functor E is a monad, as follows from a result in [22]. For a detailed discussion of this analysis of languages with explicit substitutions, see [10]. $\square$

There are also nice examples of ideal monads among computational monads. Some of these are free monads (e.g., exceptions, interactive output, interactive input monads), but others are not. In the following examples, to think of programming language semantics, assume that $\mathcal{C}$ is **Set** or **CPO**.

Example 3.6 (Exceptions)**.** Computations that raise exceptions of type E are modelled by the monad with underlying functor $\mathsf{Id} + K_E$, unit $\mathsf{inl}_{\mathsf{Id},K_E}$ and multiplication $[\mathsf{Id} + K_E, \mathsf{inr}_{\mathsf{Id},K_E}]$: a computation is either a value or an exception. This monad is ideal as it is also the free monad over the functor K_E as $\mathsf{Id} + K_E \cong \mu(\mathsf{Id} + K_E \circ \_)$. $\square$

Example 3.7 (Interactive output)**.** Computations that output tokens of type O are modelled by the monad whose underlying functor is $\mathsf{Id} \times K_{List(O)}$ where $List = \mu(K_1 + \mathsf{Id} \times \_)$. Computations are pairs of a value and an O-list, the unit pairs a value with an empty O-list and the multiplication takes a value paired with a list and then another list and returns it paired with the concatenation of the two lists. This monad is ideal as, again, it is free over the functor $K_O \times \_$ since $\mathsf{Id} \times K_{List(O)} \cong \mu(\mathsf{Id} + K_O \times \_)$. $\square$

Example 3.8 (Non-deadlocking non-determinism)**.** Non-deterministic computations that are guaranteed to not deadlock are modelled by non-empty lists of values. The corresponding functor $NEList = \mathsf{Id} \times List = \mathsf{Id} \times \mu(K_1 + \mathsf{Id} \times \_) \cong \mu(\mathsf{Id} \times (K_1 + \_))$ carries a monad structure with singleton formation and flattening a non-empty list of non-empty lists into a non-empty list as the unit and the multiplication. This monad is ideal, with singleton lists playing the role of variables and length-at-least-2 lists playing the role of non-variable terms—the multiplication restricts properly as flattening a length-at-least-2 list of non-empty lists gives a length-at-least-2 list—, but it is not free.

Note that lists, defined by $\mathit{List} = \mu(K_1 + \mathsf{Id} \times \_) \cong K_1 + \mathit{NEList}$ give a very similar monad. This monad, which models non-determinism with possible deadlocks, however, is not ideal, despite the clear separation between the variables, i.e., the singleton lists, and the non-variable terms, i.e., the empty list and the length-at-least-2 lists. The reason is that here the multiplication does not restrict as required for an ideal monad: the flattening of a length-2 list consisting of a singleton and the empty list is a singleton. □

Example 3.9 (Probabilistic choice)**.** Computations with probabilistic choice can be modelled by non-empty lists of values paired with rational numbers from $(0,1]$ that must add up to 1 over the list (these are obviously obtainable from non-empty lists of values paired with positive integers by identifying those equal under norming). Again there is a monad structure on the corresponding functor and the monad is ideal, but not free. □

In the last two examples, we could of course also have used non-empty finite multisets instead of non-empty lists, but we preferred non-empty lists here as they a definable as least fixed points and moreover they actually also provide more informative notions of computation than non-empty finite multisets.

3.2. Coproducts of Ideal Monads by a Fixed Point Construction

The fundamental observation behind the construction of the coproduct T of two ideal monads $R = \mathsf{Id} + R_0$ and $S = \mathsf{Id} + S_0$ on a category $\mathcal{C}$ is that i) T should contain as submonads R and S, and ii) T should be closed under the application of R_0 and S_0. Hence T should consist of alternating sequences beginning from R_0 or S_0. Thus we ask for the least solution to the equation system

$$T_1 \cong R_0(\mathsf{Id} + T_2) \qquad\qquad T_2 \cong S_0(\mathsf{Id} + T_1)$$

i.e., we define

$$(T_1, T_2)X = \mu(R_0(X + \_{}_2), S_0(X + \_{}_1))$$

or, equivalently, point-freely,

$$(T_1, T_2) = \mu(R_0(\mathsf{Id} + \_{}_2), S_0(\mathsf{Id} + \_{}_1))$$

and write $t_1 : R_0(\mathsf{Id} + T_2) \to T_1$, $t_2 : S_0(\mathsf{Id} + T_1) \to T_2$ for the structure maps associated to T_1, T_2. Our idea is now that $T = \mathsf{Id} + (T_1 + T_2)$. Intuitively, T_1 consists of elements in T whose top layer is a non-variable R-layer (captured by the use of R_0) and whose next layers are either variables or a non-variable S layer etc. We henceforth assume T_1 and T_2 exist, for example, we may require $\mathcal{C}$ to have ω-colimits and for R_0 and S_0 to preserve them.

Theorem 3.10 (Coproduct of two ideal monads)**.** *Let $R = \mathsf{Id}+R_0$ and $S = \mathsf{Id}+S_0$ be two ideal monads. If the functors T_1, T_2 given by the construction above are defined, then the coproduct of the ideal monads $R = \mathsf{Id}+R_0$ and $S = \mathsf{Id}+S_0$ exists and its underlying functor is $T = \mathsf{Id} + (T_1 + T_2)$.*

Proof. We first construct a candidate coproduct $((T, \eta, m), i, j)$ (a cospan in the category of monads on $\mathcal{C}$) and then we prove that it is indeed the coproduct (the initial cospan).

Construction of the candidate coproduct $((T, \eta, m), i, j)$**:** We begin by constructing the candidate coproduct carrier (T, η, m) and verifying that it is a monad. Then we proceed to constructing the candidate injections i, j and checking that they are monad morphisms.

Construction of the candidate coproduct carrier (T, η, m)*:* We have already defined T. Our candidate unit η is the injection

$$\mathsf{Id} \xrightarrow{\mathsf{inl}} \mathsf{Id} + (T_1 + T_2) = T$$

Our candidate multiplication m is

$$TT = (\mathsf{Id} + (T_1 + T_2))T \xrightarrow{[T, \mathsf{inr}\cdot(m_1+m_2)]} \mathsf{Id} + (T_1 + T_2) = T$$

where $(m_1 : T_1T \to T_1, m_2 : T_2T \to T_2)$ is the unique (by mutual iteration) pair of natural transformations such that the diagram

$$\begin{tikzcd}
R_0(\mathsf{Id} + T_2)T \arrow[rr, "t_1T"] \arrow[d, "R_0(T+m_2)"'] & & T_1T \arrow[d, "m_1"] \\
R_0(T + T_2) \arrow[r, "p_1"'] & R_0(\mathsf{Id} + T_2) \arrow[r, "t_1"'] & T_1
\end{tikzcd}$$

and

$$\begin{tikzcd}
S_0(\mathsf{Id} + T_1)T \arrow[rr, "t_2T"] \arrow[d, "S_0(T+m_1)"'] & & T_2T \arrow[d, "m_2"] \\
S_0(T + T_1) \arrow[r, "p_2"'] & S_0(\mathsf{Id} + T_1) \arrow[r, "t_2"'] & T_2
\end{tikzcd}$$

commute. Above, p_1, p_2 denote the composites

$$R_0(T + T_2) \longrightarrow R_0((\mathsf{Id} + T_2) + T_1) \xrightarrow{R_0((\mathsf{Id}+T_2)+t_1^{-1})} R_0R(\mathsf{Id} + T_2) \xrightarrow{m_0^R(\mathsf{Id}+T_2)} R_0(\mathsf{Id} + T_2)$$

$$S_0(T + T_1) \longrightarrow S_0((\mathsf{Id} + T_1) + T_2) \xrightarrow{S_0((\mathsf{Id}+T_1)+t_2^{-1})} S_0S(\mathsf{Id} + T_1) \xrightarrow{m_0^S(\mathsf{Id}+T_1)} S_0(\mathsf{Id} + T_1)$$

where the unlabelled arrows are trivial coproduct rearrangements under R_0 resp. S_0.

Proof that (T, η, m) is a monad: The post-unit law is obeyed trivially by the candidate monad (T, η, m): $m \cdot \eta T = [T, \mathsf{inr}_{\mathsf{Id},T_1+T_2} \cdot (m_1 + m_2)] \cdot \mathsf{inl}_{\mathsf{Id},T_1+T_2} T = [T, \mathsf{inr}_{\mathsf{Id},T_1+T_2} \cdot (m_1 + m_2)] \cdot \mathsf{inl}_{T,(T_1+T_2)T} = T$.

To see that the pre-unit law is met, let $(m_1^{(1)} : T_1 \to T_1, m_2^{(1)} : T_2 \to T_2)$ be the unique (by mutual iteration) pair of natural transformations such that the diagrams

$$
\begin{tikzcd}
R_0(\mathsf{Id}+T_2) \arrow[rrr, "t_1"] \arrow[d, "R_0(\mathsf{Id}+m_2^{(1)})"'] & & & T_1 \arrow[d, "m_1^{(1)}"] \\
R_0(\mathsf{Id}+T_2) \arrow[r, "R_0(\eta+T_2)"'] & R_0(T+T_2) \arrow[r, "p_1"'] & R_0(\mathsf{Id}+T_2) \arrow[r, "t_1"'] & T_1
\end{tikzcd}
$$

and

$$
\begin{tikzcd}
S_0(\mathsf{Id}+T_1) \arrow[rrr, "t_2"] \arrow[d, "S_0(\mathsf{Id}+m_1^{(1)})"'] & & & T_2 \arrow[d, "m_2^{(1)}"] \\
S_0(\mathsf{Id}+T_1) \arrow[r, "S_0(\eta+T_1)"'] & S_0(T+T_1) \arrow[r, "p_2"'] & S_0(\mathsf{Id}+T_1) \arrow[r, "t_2"'] & T_2
\end{tikzcd}
$$

commute.

Now it must be that $(m_1^{(1)}, m_2^{(1)}) = (T_1, T_2)$ as witnessed by the commuting diagram

$$
\begin{tikzcd}
R_0(\mathsf{Id}+T_2) \arrow[rrrrrr, "t_1"] \arrow[d, "R_0(\mathsf{Id}+T_2)"] & & & & & & T_1 \arrow[d, "T_1"] \\
R_0(\mathsf{Id}+T_2) \arrow[r, "R_0(\eta+T_2)"'] \arrow[rr, bend left, "R_0\mathsf{inl}"] \arrow[rrr, bend left, "R_0\eta^R(\mathsf{Id}+T_2)"] \arrow[rrrr, bend left=40, equal] & R_0(T+T_2) \arrow[r] \arrow[rrr, bend right, "p_1"'] & R_0((\mathsf{Id}+T_2)+T_1) \arrow[r] & R_0R(\mathsf{Id}+T_2) \arrow[r, "m_0^R(\mathsf{Id}+T_2)"] & R_0(\mathsf{Id}+T_2) \arrow[r, "t_1"'] & & T_1
\end{tikzcd}
$$

(in which the crucial triangle commutes by η^R being the pre-unit of m^R) and by the symmetric commuting diagram.

But it is also the case that $(m_1^{(1)}, m_2^{(1)}) = (m_1 \cdot T_1\eta, m_2 \cdot T_2\eta)$ as witnessed by the commuting diagram

$$\begin{tikzcd}
R_0(\mathsf{Id}+T_2) \arrow[r, equal] \arrow[d, "R_0(\mathsf{Id}+T_2\eta)"'] & R_0(\mathsf{Id}+T_2) \arrow[rr, "t_1"] \arrow[d, "R_0(\mathsf{Id}+T_2)\eta"] & & T_1 \arrow[d, "T_1\eta"] \\
R_0(\mathsf{Id}+T_2T) \arrow[r, "R_0(\eta+T_2T)"] \arrow[d, "R_0(\mathsf{Id}+m_2)"'] & R_0(\mathsf{Id}+T_2)T \arrow[rr, "t_1T"] \arrow[d, "R_0(T+m_2)"] & & T_1T \arrow[d, "m_1"] \\
R_0(\mathsf{Id}+T_2) \arrow[r, "R_0(\eta+T_2)"'] & R_0(T+T_2) \arrow[r, "p_1"'] & R_0(\mathsf{Id}+T_2) \arrow[r, "t_1"'] & T_1
\end{tikzcd}$$

(in which the bottom right square commutes by construction of m_1) and by the symmetric commuting diagram.

Combining the observations made, we get that $(T_1, T_2) = (m_1 \cdot T_1\eta, m_2 \cdot T_2\eta)$ and therefore $T = \mathsf{Id} + (T_1 + T_2) = [\mathsf{inl}_{\mathsf{Id},T_1+T_2}, \mathsf{inr}_{\mathsf{Id},T_1+T_2} \cdot (m_1 \cdot T_1\eta + m_2 \cdot T_2\eta)] = [T, \mathsf{inr}_{\mathsf{Id},T_1+T_2} \cdot (m_1 + m_2)] \cdot (\mathsf{Id} + (T_1 + T_2))\eta = m \cdot T\eta$.

To verify associativity of multiplication, let $(m_1^{(3)} : T_1TT \to T_1, m_2^{(3)} : T_2TT \to T_2)$ be the unique (by mutual iteration) pair of natural transformations such that diagrams

$$\begin{tikzcd}
R_0(\mathsf{Id}+T_2)TT \arrow[rrr, "t_1TT"] \arrow[d, "R_0(TT+m_2^{(3)})"'] & & & T_1TT \arrow[d, "m_1^{(3)}"] \\
R_0(TT+T_2) \arrow[r, "R_0(m+T_2)"'] & R_0(T+T_2) \arrow[r, "p_1"'] & R_0(\mathsf{Id}+T_2) \arrow[r, "t_1"'] & T_1
\end{tikzcd}$$

and

$$\begin{tikzcd}
S_0(\mathsf{Id}+T_1)TT \arrow[rrr, "t_2TT"] \arrow[d, "S_0(TT+m_1^{(3)})"'] & & & T_2TT \arrow[d, "m_2^{(3)}"] \\
S_0(TT+T_1) \arrow[r, "S_0(m+T_1)"'] & S_0(T+T_1) \arrow[r, "p_2"'] & S_0(\mathsf{Id}+T_1) \arrow[r, "t_2"'] & T_2
\end{tikzcd}$$

commute.

We get $(m_1^{(3)}, m_2^{(3)}) = (m_1 \cdot m_1T, m_2 \cdot m_2T)$ from the commuting diagram

$$\begin{tikzcd}
R_0(\mathsf{Id}+T_2)TT \arrow[rrrrr, "t_1TT"] \arrow[d, "R_0(T+m_2)T"] & & & & & T_1TT \arrow[d, "m_1T"] \\
R_0(T+T_2)T \arrow[r, equal] \arrow[ddd, "R_0(TT+m_2)"] & R_0(T+T_2)T \arrow[r] \arrow[ddd, "R_0(m+m_2)"] & R_0((\mathsf{Id}+T_2)+T_1)T \arrow[r, "R_0((\mathsf{Id}+T_2)+t_1^{-1})T"] & R_0R(\mathsf{Id}+T_2)T \arrow[r, "m_0^R(\mathsf{Id}+T_2)T"] \arrow[d, "R_0R(T+m_2)"] & R_0(\mathsf{Id}+T_2)T \arrow[r, "t_1T"] \arrow[d, "R_0(T+m_2)"] & T_1T \arrow[ddd, "m_1"] \\
& & & R_0R(T+T_2) \arrow[r, "m_0^R(T+T_2)"] \arrow[d] & R_0(T+T_2) \arrow[d] & \\
& & & R_0RR(\mathsf{Id}+T_2) \arrow[r, "m_0^R R(\mathsf{Id}+T_2)"] \arrow[d, "R_0m^R(\mathsf{Id}+T_2)"] & R_0R(\mathsf{Id}+T_2) \arrow[d, "m_0^R(\mathsf{Id}+T_2)"] & \\
R_0(TT+T_2) \arrow[r, "R_0(m+T_2)"'] & R_0(T+T_2) \arrow[r] \arrow[rrr, bend right, "p_1"'] & R_0((\mathsf{Id}+T_2)+T_1) \arrow[r, "R_0((\mathsf{Id}+T_2)+t_1^{-1})"] & R_0R(\mathsf{Id}+T_2) \arrow[r, "m_0^R(\mathsf{Id}+T_2)"] & R_0(\mathsf{Id}+T_2) \arrow[r, "t_1"'] & T_1
\end{tikzcd}$$

and the symmetric diagram. In the diagram shown, the upper half and the 2nd and 4th quarters in the lower half commute by construction of m_1, and the closest to the bottom small square in the lower half commutes by associativity of m^R.

And we also get $(m_1^{(3)}, m_2^{(3)}) = (m_1 \cdot T_1 m, m_2 \cdot T_2 m)$ from the commuting diagram

$$\begin{tikzcd}
R_0(\mathsf{Id}+T_2)TT \arrow[r, equal] \arrow[d, "R_0(TT+T_2m)"'] & R_0(\mathsf{Id}+T_2)TT \arrow[rr, "t_1TT"] \arrow[d, "R_0(\mathsf{Id}+T_2)m"] & & T_1TT \arrow[d, "T_1m"] \\
R_0(T+T_2)T \arrow[r, "R_0(m+T_2T)"] \arrow[d, "R_0(TT+m_2)"'] & R_0(\mathsf{Id}+T_2)T \arrow[rr, "t_1T"] \arrow[d, "R_0(T+m_2)"] & & T_1T \arrow[d, "m_1"] \\
R_0(TT+T_2) \arrow[r, "R_0(m+T_2)"'] & R_0(T+T_2) \arrow[r, "p_1"'] & R_0(\mathsf{Id}+T_2) \arrow[r, "t_1"'] & T_1
\end{tikzcd}$$

and the symmetric diagram. In the diagram given, the bottom right square commutes by construction of m_1.

Putting these pieces of knowledge together, we get that $(m_1 \cdot m_1T, m_2 \cdot m_2T) = (m_1 \cdot T_1 m, m_2 \cdot T_2 m)$ and hence $m \cdot mT = [T, \mathsf{inr} \cdot (m_1 + m_2)] \cdot [T, \mathsf{inr} \cdot (m_1 + m_2)]T = [m, \mathsf{inr} \cdot (m_1 + m_2) \cdot (m_1 + m_2)T] = [m, \mathsf{inr} \cdot (m_1 + m_2) \cdot (T_1 + T_2)m] = [T, \mathsf{inr} \cdot (m_1 + m_2)] \cdot (\mathsf{Id} + (T_1 + T_2))m = m \cdot Tm$.

We realize that in addition to (T, η, m) being a monad the data $(T, \eta, m, T_1 + T_2, m_1 + m_2)$ form an ideal monad.

Construction of the candidate injections i, j: Next, we need monad morphisms $i : R \to T$, $j : S \to T$ to play the role of injections. We propose to choose

$i = \mathsf{Id} + (\mathsf{inl}_{T_1,T_2} \cdot i_0)$, $j = \mathsf{Id} + (\mathsf{inr}_{T_1,T_2} \cdot j_0)$ where i_0, j_0 are the composites

$$R_0 \xrightarrow{R_0\mathsf{inl}} R_0(\mathsf{Id}+T_2) \xrightarrow{t_1} T_1 \qquad\qquad S_0 \xrightarrow{S_0\mathsf{inl}} S_0(\mathsf{Id}+T_1) \xrightarrow{t_2} T_2$$

Proof that i, j are monad morphisms: Preservation of the unit by i is obvious: $i \cdot \eta^R = (\mathsf{Id} + (\mathsf{inl}_{T_1,T_2} \cdot i_0)) \cdot \mathsf{inl}_{\mathsf{Id},R_0} = \mathsf{inl}_{\mathsf{Id},T_1+T_2} = \eta$.

To prove that i preserves the multiplication, we first check that $i_0 \cdot m_0^R = m_1 \cdot i_0T \cdot R_0 i$ from the commuting diagram

$$\begin{tikzcd}
 & R_0R \arrow[r, "m_0^R"] \arrow[dl, "R_0(\mathsf{Id}+R_0\mathsf{inl})"'] \arrow[d, "R_0R\mathsf{inl}"] & R_0 \arrow[d, "R_0\mathsf{inl}"] \\
R_0(\mathsf{Id}+R_0(\mathsf{Id}+T_2)) \arrow[r, "R_0(\mathsf{inl}+R_0(\mathsf{Id}+T_2))"'] \arrow[d, "R_0(\mathsf{Id}+t_1)"'] & R_0R(\mathsf{Id}+T_2) \arrow[r, "m_0^R(\mathsf{Id}+T_2)"] \arrow[d, bend right, "R_0((\mathsf{Id}+T_2)+t_1)"'] & R_0(\mathsf{Id}+T_2) \arrow[dddd, "t_1"] \\
R_0(\mathsf{Id}+T_1) \arrow[r, "R_0(\mathsf{inl}+T_1)"] \arrow[d, "R_0(\mathsf{Id}+\mathsf{inl})"'] & R_0((\mathsf{Id}+T_2)+T_1) \arrow[u, bend right, "R_0((\mathsf{Id}+T_2)+t_1^{-1})"'] & \\
R_0T \arrow[r, "R_0\mathsf{inl}"] \arrow[dr, "R_0\mathsf{inl}T"'] & R_0(T+T_2) \arrow[u] \arrow[uur, "p_1"'] & \\
 & R_0(\mathsf{Id}+T_2)T \arrow[u, "R_0(T+m_2)"'] \arrow[d, "t_1T"'] & \\
 & T_1T \arrow[r, "m_1"'] & T_1
\end{tikzcd}$$

where the bottom right square commutes by construction of m_1. From this basis we obtain that $i \cdot m^R = (\mathsf{Id} + (\mathsf{inl}_{T_1,T_2} \cdot i_0)) \cdot [R, \mathsf{inr} \cdot m_0^R] = [i, \mathsf{inr} \cdot \mathsf{inl}_{T_1,T_2} \cdot i_0 \cdot m_0^R] = [i, \mathsf{inr} \cdot \mathsf{inl}_{T_1,T_2} \cdot m_1 \cdot i_0T \cdot R_0 i] = [T, \mathsf{inr} \cdot (m_1 + m_2)] \cdot (\mathsf{Id} + (\mathsf{inl}_{T_1,T_2} \cdot i_0))T \cdot (\mathsf{Id} + R_0)i = m \cdot iT \cdot Ri$.

That j also preserves the unit and the multiplication is proved symmetrically.

Proof that $((T, \eta, m), i, j)$ is the coproduct: We now turn to the proof that that $((T, \eta, m), i, j)$ really is a coproduct, i.e., to the construction of copairing.

Construction of the candidate mediating morphism h to given (H, f, g): Given a monad H and two monad morphisms $f = [\eta^H, f_0] : R \to H$, $g = [\eta^H, g_0] : S \to H$ induced by $f_0, g_0 : R_0, S_0 \to H$, let $(h_1 : T_1 \to H, h_2 : T_2 \to H)$ be the unique (by mutual iteration) pair of natural transformations such that the diagrams

$$
\begin{tikzcd}
R_0(\mathsf{Id}+T_2) \arrow[rr, "t_1"] \arrow[d, "R_0(\mathsf{Id}+h_2)"] & & T_1 \arrow[d, "h_1"] \\
R_0(\mathsf{Id}+H) \arrow[r, "{f_0[\eta^H, H]}"'] & HH \arrow[r, "m^H"'] & H
\end{tikzcd}
\quad \text{and} \quad
\begin{tikzcd}
S_0(\mathsf{Id}+T_1) \arrow[rr, "t_1"] \arrow[d, "S_0(\mathsf{Id}+h_1)"] & & T_2 \arrow[d, "h_1"] \\
S_0(\mathsf{Id}+H) \arrow[r, "{g_0[\eta^H, H]}"'] & HH \arrow[r, "m^H"'] & H
\end{tikzcd}
$$

commute. Our claim is that $h = [\eta^H, [h_1, h_2]] : T \to H$ is the copair of f, g. We have to show that h is a monad morphism, a mediating morphism and the unique such.

Proof that h is a monad morphism: Preservation of the unit by h is trivial: $h \cdot \eta = [\eta^H, [h_1, h_2]] \cdot \mathsf{inl}_{\mathsf{Id}, T_1+T_2} = \eta^H$.

To show that h preserves the multiplication, let $(k_1 : T_1T \to H, k_2 : T_2T \to H)$ be the unique (by mutual iteration) pair of natural transformations such that the diagrams

$$
\begin{tikzcd}
R_0(\mathsf{Id}+T_2)T \arrow[rr, "t_1 T"] \arrow[d, "R_0(T+k_2)"] & & T_1T \arrow[d, "k_1"] \\
R_0(T+H) \arrow[r, "{f_0[h, H]}"'] & HH \arrow[r, "m^H"'] & H
\end{tikzcd}
\quad \text{and} \quad
\begin{tikzcd}
S_0(\mathsf{Id}+T_1)T \arrow[rr, "t_2 T"] \arrow[d, "S_0(T+k_1)"] & & T_2T \arrow[d, "k_2"] \\
S_0(T+H) \arrow[r, "{g_0[h, H]}"'] & HH \arrow[r, "m^H"'] & H
\end{tikzcd}
$$

commute.

Now $(k_1, k_2) = (h_1 \cdot m_1, h_2 \cdot m_2)$ as the commuting diagram

$$
\begin{tikzcd}
R_0(\mathsf{Id}+T_2)T \arrow[rrrrr, "t_1 T"] \arrow[d, "R_0(T+m_2)"] & & & & & T_1T \arrow[d, "m_1"] \\
R_0(T+T_2) \arrow[r] \arrow[dddd, "R_0(T+h_2)"] & R_0((\mathsf{Id}+T_2)+T_1) \arrow[r, "R_0((\mathsf{Id}+T_2)+t_1^{-1})"] \arrow[dddd, "{R_0[[\eta^H, h_2], h_1]}"] & R_0R(\mathsf{Id}+T_2) \arrow[rr, "m_0^R(\mathsf{Id}+T_2)"] \arrow[d, "{R_0R[\eta^H, h_2]}"] & & R_0(\mathsf{Id}+T_2) \arrow[r, "t_1"] \arrow[d, "{R_0[\eta^H, h_2]}"] & T_1 \arrow[dddd, "h_1"] \\
& & R_0RH \arrow[rr, "m_0^R H"] \arrow[d, "{R_0[\eta^H, f_0]H}"] & & R_0H \arrow[d, "f_0 H"] & \\
& & R_0HH \arrow[r, "f_0HH"] \arrow[dd, "R_0 m^H"] & HHH \arrow[r, "m^H H"] \arrow[dd, "H m^H"] & HH \arrow[dd, "m^H"] & \\
& & & & & \\
R_0(T+H) \arrow[r, "{R_0[h, H]}"'] & R_0H \arrow[r, equal] & R_0H \arrow[r, "f_0 H"'] & HH \arrow[r, "m^H"'] & H \arrow[r, equal] & H
\end{tikzcd}
$$

and the symmetric diagram demonstrate. In the diagram given, the upper half commutes by construction of m_1. In the lower half, the 2nd and 4th quarters commute by construction of h_1, the middle third of the 3rd quarter commutes by preservation of the multiplication by $f = [\eta^H, f_0]$ and the rightmost smallest square by associativity of m^H.

But we can also get $(k_1, k_2) = (m^H \cdot h_1 H \cdot T_1 h, m^H \cdot h_2 H \cdot T_2 h)$ as is evident from the commuting diagram

$$\begin{tikzcd}
R_0(\mathsf{Id}+T_2)T \arrow[rrrr, "t_1 T"] \arrow[d, "R_0(T+T_2h)"'] & & & & T_1 T \arrow[d, "T_1 h"] \\
R_0(T+T_2H) \arrow[r, "R_0(h+T_2H)"] \arrow[d, "R_0(T+h_2H)"'] & R_0(\mathsf{Id}+T_2)H \arrow[rrr, "t_1 H"] \arrow[d, "R_0(\mathsf{Id}+h_2)H"] & & & T_1 H \arrow[d, "h_1 H"] \\
R_0(T+HH) \arrow[r, "R_0(h+HH)"] \arrow[d, "R_0(T+m^H)"'] & R_0(\mathsf{Id}+H)H \arrow[r, "R_0[\eta^H, H]H"] \arrow[d, "R_0[H, m^H]"] & R_0 HH \arrow[r, "f_0 HH"] \arrow[d, "R_0 m^H"] & HHH \arrow[r, "m^H H"] \arrow[d, "H m^H"] & HH \arrow[d, "m^H"] \\
R_0(T+H) \arrow[r, "R_0(h+H)"'] & R_0 H \arrow[r, equals] & R_0 H \arrow[r, "f_0 H"'] & HH \arrow[r, "m^H"'] & H
\end{tikzcd}$$

and the symmetric diagram. In the diagram shown, the 2nd half of the middle third commutes by construction of h_1 and the 2nd and 4th quarters of the lower third commute by η^H being the pre-unit of m^H resp. by associativity of m^H.

As a consequence $(h_1 \cdot m_1, h_2 \cdot m_2) = (m^H \cdot h_1 H \cdot T_1 h, m^H \cdot h_2 H \cdot T_2 h)$ and therefore $h \cdot m = [\eta^H, [h_1, h_2]] \cdot [T, \mathsf{inr}_{\mathsf{Id}, T_1+T_2} \cdot (m_1 + m_2)] = [h, [h_1 \cdot m_1, h_2 \cdot m_2]] = [m^H \cdot \eta^H H \cdot h, [m^H \cdot h_1 H \cdot T_1 h, m^H \cdot h_2 H \cdot T_2 h]] = m^H \cdot [\eta^H, [h_1, h_2]]H \cdot (\mathsf{Id} + (T_1 + T_2))h = m^H \cdot hH \cdot Th$.

Proof that h is a mediating morphism: To prove that $h \cdot i = f$, we first record that $h_1 \cdot i_0 = f_0$ from the commuting diagram

$$\begin{tikzcd}
R_0 \arrow[r, "R_0\mathsf{inl}"] \arrow[rrrr, bend left, "i_0"] \arrow[dr, "R_0\mathsf{inl}"'] \arrow[ddrr, equals, bend right] & R_0(\mathsf{Id}+T_2) \arrow[rrr, "t_1"] \arrow[d, "R_0(\mathsf{Id}+h_2)"] & & & T_1 \arrow[d, "h_1"] \\
& R_0(\mathsf{Id}+H) \arrow[r, "R_0[\eta^H, H]"] & R_0 H \arrow[r, "f_0 H"] & HH \arrow[r, "m^H"] & H \\
& & R \arrow[u, "R_0\eta^H"] \arrow[r, "f_0"'] & H \arrow[u, "H\eta^H"] \arrow[ur, equals] &
\end{tikzcd}$$

which uses the construction of h_1 and the fact that η^H is a pre-unit of m^H. Further we get that $h \cdot i = [\eta^H, [h_1, h_2]] \cdot (\mathsf{Id} + (\mathsf{inl}_{T_1, T_2} \cdot i_0)) = [\eta^H, h_1 \cdot i_0] = [\eta^H, f_0] = f$.

The proof that $h \cdot j = g$ is symmetric.

Proof that h is the only mediating morphism Uniqueness of h in the capacity of a morphism mediating between (T, i, j) and (H, f, g) is proved by the following commuting diagram and the symmetric diagram, which show that from $h' =$

$[\eta^H, [h'_1, h'_2]]$ being a mediator it follows by construction of (h_1, h_2) that $(h'_1, h'_2) = (h_1, h_2)$. From this, the desired result $h' = h$ is immediate.

$$
\begin{tikzcd}
 & & R_0R(\mathsf{Id}+T_2) \arrow[r, "m_0^R(\mathsf{Id}+T_2)"'] & R_0(\mathsf{Id}+T_2) \arrow[dddd, "t_1"] \\
 & & R_0((\mathsf{Id}+T_2)+T_1) \arrow[u] & \\
 & & R_0(T+T_2) \arrow[u] & \\
R_0(\mathsf{Id}+T_2) \arrow[uuurrr, equal, bend left] \arrow[uuurr, "R_0\eta^R(\mathsf{Id}+T_2)"] \arrow[uurr, "R_0\mathsf{inl}"] \arrow[r, "R_0(\mathsf{Id}+\mathsf{inr})"] \arrow[ddr, "f_0(\mathsf{Id}+T_2)"'] & R_0T \arrow[ur, "R_0\mathsf{inl}"] \arrow[r, "R_0\mathsf{inl}T"] \arrow[dr, "i_0"] \arrow[ddr, "f_0T"'] & R_0(\mathsf{Id}+T_2)T \arrow[u, "R_0(T+m_2)"'] \arrow[d, "t_1T"] & \\
 & & T_1T \arrow[r, "m_1"] \arrow[d, "h'_1T"] & T_1 \arrow[dd, "h'_1"'] \\
 & H(\mathsf{Id}+T_2) \arrow[r, "H(\mathsf{Id}+\mathsf{inr})"] \arrow[dr, "{H[\eta^H, h'_2]}"'] & HT \arrow[d, "{H[\eta^H,[h'_1,h'_2]]}"] & \\
 & & HH \arrow[r, "m^H"'] & H
\end{tikzcd}
$$

The top right square of the diagram commutes by construction of t_1, the bottom right square by h' preserving the multiplication, the top triangle by η^R being a pre-unit of m^R. One of the triangles in the middle reflects the assumption that $f \cdot i = h'$.

Intuitively, uniqueness should not be surprising since any mediating monad morphism $h' : T \to H$ must equal $h = [\eta^H, [h_1, h_2]]$ i) on variables because monad morphisms preserve the unit; ii) on R_0 and S_0 because on them it is determined by f_0, g_0; and iii) on all other elements of T since they are essentially multiplications of R_0 and S_0 which are preserved by monad morphisms. $\square$

Using Bekić lemma, the definition of T from the theorem can be rewritten without the use of a simultaneous fixed point: by the lemma,

$$
\begin{aligned}
T_1 &\cong \mu(R_0(\mathsf{Id} + S_0(\mathsf{Id} + \_))) \\
T_2 &\cong S_0(\mathsf{Id} + T_1)
\end{aligned}
$$

and hence (by the rolling lemma in the last step)

$$\begin{array}{rcl} T & = & \mathsf{Id} + (T_1 + T_2) \\ & \cong & (\mathsf{Id} + T_1) + T_2 \\ & \cong & (\mathsf{Id} + T_1) + S_0(\mathsf{Id} + T_1) \\ & = & S(\mathsf{Id} + T_1) \\ & \cong & S(\mathsf{Id} + \mu(R_0(\mathsf{Id} + S_0(\mathsf{Id} + \_)))) \\ & \cong & S(\mu(\mathsf{Id} + R_0(\mathsf{Id} + S_0 \circ \_))) \end{array}$$

4. Applications

Having at our disposal a general construction for the coproduct of two ideal monads (Theorem 3.10), we can apply it to our example ideal monads, especially to those that are not free and thus out of the reach of the more specific result on the coproduct of a free monad with any monad (Proposition 2.9). We now turn to presenting some such applications.

Example 4.1 (Coproduct of two free completely iterative monads)**.** We know that the coproduct of two free monads $T_F = \mu(\mathsf{Id} + F \circ \_)$ and $T_G = \mu(\mathsf{Id} + G \circ \_)$ is $T_{F+G} = \mu(\mathsf{Id} + (F+G) \circ \_)$. This might provoke us to conjecture that the coproduct of two free completely iterative monads $T_F^\infty = \nu(\mathsf{Id} + F \circ \_)$ and $T_G^\infty = \nu(\mathsf{Id} + G \circ \_)$ is $T_{(F+G)}^\infty = \nu(\mathsf{Id} + (F+G) \circ \_)$. Theorem 3.10 tells us however, that this cannot be: we get finite interleavings of FT_F^∞ and GT_G^∞ which means that, e.g., infinite alternations of F and G are not in the coproduct.

Intuitively, T_{F+G}^∞ fails to be the coproduct because it is too large. The coproduct must be the least functor containing T_F^∞ and T_G^∞ and closed under variables and substitution. To produce such a functor "from below", one need not start with T_F^∞ and T_G^∞ and then allow infinite combinations of these with substitution: it is clearly enough to allow substitution to be used only a finite number of times. □

Example 4.2 (Binding operators cannot be added with a coproduct)**.** Our central theorem allows us to consider questions such as the modularity of higher order syntax. For example, one would expect the language of the lambda-calculus $L = \mu(\mathsf{Id} + \Delta \circ \_ + \_ \circ \delta)$ to be the coproduct of $L_{\text{app}} = \mu(\mathsf{Id} + \Delta \circ \_) \cong \mathsf{Id} + \Delta L_{\text{app}}$ and $L_{\text{abs}} = \mu(\mathsf{Id} + \_ \circ \delta) \cong \mathsf{Id} + L_{\text{abs}}\delta$ since this kind of modularity holds for first order terms. However, this is not the case as can be deduced from Theorem 3.10: the coproduct consists of all finite interleavings of ΔL_{app} and $L_{\text{abs}}\delta$ and that, sadly enough, excludes even the term $\lambda y.xy$: the body of λ-abstraction can only be an application if the application merely uses the context from outside the λ-abstraction: terms $\lambda y.st$ where y is in the context of s or t are forbidden. The reason is that terms such as $\lambda y.xy$ are not constructible from applications and λ-abstractions using variables and (the proper, capture-avoiding) substitution only, one needs the naive substitution to manufacture $\lambda y.xy$. A closer account of this phenomenon is given in [10]. □

Example 4.3 (Combining non-deadlocking non-determinism with probabilistic choice)**.** What the right way is to combine non-determinism with probabilistic choice has been a long-standing question in programming language semantics, cf. [13,23,27,28]. Notably, for example, there is no distributive law in **Set** of the non-determinism monad over the probabilistic choice monad and none in the opposite direction either[1]. Theorem 3.10 tells us that the coproduct consists of all finite interleavings of non-trivial non-determinism and non-trivial probabilistic choice: one can think of its elements as wellfounded trees whose leaves are labelled by values and whose branching nodes have branching factor at least two and have their outgoing nodes alternately either unlabelled (corresponding to non-determinism) or labelled with rational numbers from $(0, 1]$ adding up to 1 (corresponding to probabilistic choice). This datatype is very close to the form of synchronization trees used by Varacca [28], except that he allows trivial branchings. □

We feel that these example applications we have shown are very encouraging and there is much that can be concluded about combinability of monads using Theorem 3.10 especially in relation to non-free monads relevant for notions of syntax or computation.

5. Conclusions and Future Work

The central thesis of this paper is that i) monads model a number of important and interesting syntactic and computational phenomena; ii) understanding how to combine monads can give us information about how these phenomena interact; and iii) coproducts of monads are the canonical mechanism for combining monads. Current techniques for calculating the coproduct of monads tend to be either general and suffer from resulting complexity or specialised in which case they suffer from non-applicability to a number of important situations. We have addressed this problem by showing how the coproduct of a large number of monads, including many where research is currently very active, can be reduced to fixed point formulae involving only functorial operations. The applicability of these results is demonstrated by Section 4.

More generally, we feel that the results contained in this paper are just what is required to spur research in the area of modular syntax and computation. We are currently working on two such. Firstly, we have seen in Section 4 a manifestation of a form of non-modularity of higher-order syntax. But from general principles there must be modularity! Surely what this result highlights is that one simply needs to take the coproduct of monads in a different category. Our second idea concerns modular proof theory. Clearly theorems like cut-elimination for various simply typed λ-calculi are modular and yet there is no mathematical mechanism for formally expressing this fact. By viewing a simply typed λ-calculus as a monad over the category of graphs, cut-elimination becomes the existence of a distributive law over the path functor. We can then give a modular, and semantic, proof of

[1]Varacca [28] claims this referring to Plotkin, private communication.

cut-elimination by proving the coproduct of monads with such distributive laws also has a distributive law, i.e., also has a cut-elimination theorem.

It is also worth finishing this paper by contrasting our approach with that of Hyland, Plotkin and Power [12] where the object of study is not monads but rather other abstractions, e.g., (enriched) Lawvere theories, and other combinators are also considered, e.g., the tensor product and a distributive combination. We certainly are very interested in these other combinators and plan to investigate their application in due course. As for whether monads really provide the right level of abstraction or whether there is finer structure at play, this is a question of active research and it is too early for a definitive answer. While there is certainly merit in the argument that the use of many monads is really dependent on the operations and equations defining them, in our work on modular rewriting [20] we found that the key properties such as confluence and layer structure were much easier to state at the level of monads than at, say, the level of presentations or classifying categories. Our overall view is thus pragmatic—the results and approach of this paper seem appropriate, for our needs, but deeper and more fundamental results may well change this view.

The authors thank the referees and John Power for their insightful comments. N. Ghani's research was supported by EPSRC under grant No. GR/M96230/01 and by the Royal Society of London under the grant *Coalgebra and Recursion*. T. Uustalu's research was supported by the Estonian Science Foundation under grant No. 5567 and his participation at ETAPS 2003 was made possible by a travel grant from the Estonian Information Technology Foundation.

References

[1] P. Aczel, J. Adámek, S. Milius, and J. Velebil, Infinite trees and completely iterative theories: a coalgebraic view, *Theor. Comp. Sci.* **300**(1–3) (2003) 1–45.

[2] J. Adámek, S. Milius, and J. Velebil, Free iterative theories: a coalgebraic view, *Math. Struct. in Comp. Sci.* **13**(2) (2003) 259–320.

[3] J. Adámek and J. Rosický, *Locally Presentable and Accessible Categories*, *London Math. Soc. Lect. Note Series* **189**, Cambridge University Press, Cambridge (1994).

[4] M. Barr, Coequalizers and free triples, *Math. Z.* **116** (1970) 307–322.

[5] M. Barr and C. Wells, *Toposes, Triples and Theories*, *Grundlehren der mathematischen Wissenschaften* **275**, Springer-Verlag, Berlin (1985).

[6] J. Beck, Distributive laws, in *Seminar on Triples and Categorical Homology Theory (ETH, 1966/67)*, edited by B. Eckmann, *Lect. Notes in Math.* **80**, Springer-Verlag, Berlin (1969) 119–140.

[7] M. Fiore, G. Plotkin, and D. Turi. Abstract syntax and variable binding, in *Proc. of 14th Ann. IEEE Symp. on Logic in Computer Science, LICS'99 (Trento, July 1999)*, IEEE CS Press, Los Alamitos, CA (1999) 193–202.

[8] N. Ghani, C. Lüth, F. de Marchi, and J. Power, Dualising initial algebras, *Math. Struct. in Comp. Sci.* **13**(2) (2003) 349–370.

[9] N. Ghani, C. Lüth, and F. De Marchi, Coalgebraic monads, in *Proc. of 5th Wksh. on Coalgebraic Methods in Computer Science, CMCS'02 (Grenoble, Apr. 2002)*, edited by L. S. Moss, *Electr. Notes in Theor. Comp. Sci.* **65**(1), Elsevier, Amsterdam (2002).

[10] N. Ghani and T. Uustalu, Explicit substitutions and higher-order syntax, in *Proc. of 2nd ACM SIGPLAN Wksh. on Mechanized Reasoning about Languages with Variable Binding, MERLIN'03 (Uppsala, Aug. 2003)*, edited by F. Honsell, M. Miculan, A. Momigliano, ACM Press, New York (2003).

[11] J. A. Goguen, A categorical manifesto, in *Math. Struct. in Comp. Sci.* **1**(1) (1991) 49–67.

[12] M. Hyland, G. Plotkin, and J. Power, Combining computational effects: commutativity and sum, in *Proc. of IFIP 17th World Computer Congress, TC1 Stream / 2nd IFIP Int. Conf. on Theoretical Computer Science, TCS 2002 (Montreal, Aug. 2002)*, edited by A. Baeza-Yates, U. Montanari, and N. Santoro, *IFIP Conf. Proc.* **223**, Kluwer Academic Publishers, Dordrecht (2002) 474–484.

[13] C. Jones and G. D. Plotkin, A probabilistic powerdomain of evaluations, in *Proc. of 4th Ann. IEEE Symp. Logic in Computer Science, LICS'89 (Pacific Grove, CA, June 1989)*, IEEE CS Press, Washington, DC (1989) 186–195.

[14] M. Jones and L. Duponcheel, Composing monads, Techn. report RR-1004, Dept. of Comp. Sci, Yale Univ. (1993).

[15] G. M. Kelly, A unified treatment of transfinite constructions for free algebras, free monoids, colimits, associated sheaves and so on, *Bull. of Australian Mathematical Society* **22** (1980) 1–83.

[16] G. M. Kelly and J. Power, Adjunctions whose counits are equalizers, and presentations of finitary monads, *J. of Pure and Applied Algebra* **89** (1993) 163–179.

[17] F. E. J. Linton, Some aspects of equational categories, in *Proc. of Conf. on Categorical Algebra (La Jolla, CA, June 1965)*, edited by S. Eilenberg, D. K. Harrison, S. Mac Lane, H. Röhrl, Springer-Verlag, Berlin (1966) 84–94.

[18] C. Lüth, *Categorical Term Rewriting: Monads and Modularity*, PhD thesis, Lab. for Foundations of Comp. Sci., Univ. of Edinburgh (1998).

[19] C. Lüth and N. Ghani, Monads and modularity, in *Proc. of 4th Int. Wksh. on Frontiers of Combining Systems, FroCoS 2002 (Santa Margherita Ligure, Apr. 2002)*, edited by A. Armando, *Lect. Notes in Artif. Intell.* **2309**, Springer-Verlag, Berlin (2002) 18–32.

[20] C. Lüth and N. Ghani, Monads and modular term rewriting, in *Proc. of 7th Int. Conf. on Category Theory in Computer Science, CTCS'97 (Santa Margherita Ligure, Sept. 2002)*, edited by E. Moggi and G. Rosolini, *Lect. Notes in Comp. Sci.* **1290**, Springer-Verlag, Berlin (1997) 69–86.

[21] E. G. Manes, *Algebraic Theories*, *Graduate Texts in Mathematics* **26**, Springer-Verlag, Berlin, 1976.

[22] R. Matthes and T. Uustalu, Substitution in non-wellfounded syntax with variable binding, in *Proc. of 6th Wksh. on Coalgebraic Methods in Computer Science, CMCS'03 (Warsaw, Apr. 2003)*, edited by H. P. Gumm, *Electr. Notes in Theor. Comp. Sci.* **82**(1), Elsevier, Amsterdam (2003).

[23] M. Mislove, Nondeterminism and probabilistic choice: obeying the laws, in *Proc. 11 Int. Conf. on Concurrency Theory, CONCUR 2000 (University Park, PA, Aug. 2000)*, edited by C. Palamidessi, *Lecture Notes in Comp. Sci.* **1877**, Springer-Verlag, Berlin (2000) 350–364.

[24] E. Moggi, Computational lambda-calculus and monads, in *Proc. of 4th Ann. IEEE Symp. on Logic in Computer Science, LICS'89 (Pacific Grove, CA, June 1989)*, IEEE CS Press, Washington, DC (1989) 14–23.

[25] E. Moggi, An abstract view of programming languages, Techn. report ECS-LFCS-90-113, Lab. for Foundations of Comp. Sci., Univ. of Edinburgh (1990).

[26] L. Moss, Parametric corecursion, *Theor. Comp. Sci.* **260**(1–2) (2001) 139–163.

[27] R. Tix, Continuous D-cones: convexity and powerdomain constructions, PhD thesis, Techn. Univ. Darmstadt (1999).

[28] D. Varacca, The powerdomain of indexed valuations, in *Proc. of 17th Ann. IEEE Symp. on Logic in Computer Science, LICS'02 (Copenhagen, July 2002)*, IEEE CS Press, Los Alamitos, CA (2002) 299–308.

Communicated by (The editor will be set by the publisher).
Received 20 October 2003, accepted 18 February 2004, final version 4 May 2004.