The Coinductive Resumption Monad

Maciej Piróg[1], Jeremy Gibbons[2]

Department of Computer Science
University of Oxford

Abstract

Resumptions appear in many forms as a convenient abstraction, such as in semantics of concurrency and as a programming pattern. In this paper we introduce generalised resumptions in a category-theoretic, coalgebraic context and show their basic properties: they form a monad, they come equipped with a corecursion scheme in the sense of Adámek *et al.*'s notion of completely iterative monads (cims), and they enjoy a certain universal property, which specialises to the coproduct with a free cim in the category of cims.

Keywords: resumptions, completely iterative monads, coalgebra

1 Introduction

1.1 Resumptions

Resumptions were introduced by Milner [32] to denote the external behaviour of a communicating agent in concurrency theory. In categorical terms, as given by Abramsky [1], a resumption is an element of the carrier of the final coalgebra νR of the functor $RX = (X \times O)^I$ on Set, where I and O are the sets of possible inputs and outputs respectively. Informally, a resumption is a function that consumes some input and returns some output paired with another resumption, and finality implies that the process of consuming and producing can be iterated indefinitely. There are many possible generalisations, for example to the final coalgebra of the functor $\mathcal{P}^{\text{fin}}((\text{-}) \times O)^I$ for agents with finitely-branching nondeterministic behaviour, or, depending on

[1] maciej.pirog@cs.ox.ac.uk
[2] jeremy.gibbons@cs.ox.ac.uk

the computational effect realised by the agent, any monad in place of $\mathcal{P}^{\text{fin}}$, as studied by Hasuo and Jacobs [24].

The idea of 'resumable' computations appeared also in the context of computational effects and monadic programming. Cenciarelli and Moggi [13] defined what they called the generalised resumption monad transformer as $TA = \mu X.M(\Sigma X + A)$, where M is a monad, Σ is an endofunctor, and A is an object of variables, which allows to sequentially compose resumptions. The resumption monad can be alternatively given by $M(\Sigma M)^*$, a composition of M and the free monad generated by the composition of Σ and M. It is canonical in the sense that it is the coproduct of M and Σ^* in the category of monads and monad morphisms, as shown by Hyland, Plotkin, and Power [26].

The resumption monad is given by an initial algebra, so it is not exactly a generalisation of the resumptions studied by Milner and others. Intuitively, it models resumptions that can be iterated only finitely many times. Thus, it is natural to ask about final coalgebras of functors of the shape $M(\Sigma(\text{-}) + A)$. Indeed, Goncharov and Schröder [21] used monads of the shape $TA = \nu X.M(X + A)$ to give semantics to concurrent processes with generic effects, while the monad $TA = \nu X.M(\Sigma X + A)$ was discussed by the present authors [39] under the name "coinductive resumption monad". In this paper, we further generalise the latter construction and take a closer look at its properties.

1.2 Coinduction

Usually, an effect-free data structure is called coinductive if it is given by the carrier of a final coalgebra. Informally, finality means that a coalgebra $c : X \to FX$ that describes one step can be repeated indefinitely to build a structure of type νF layer by layer. In the monadic world, however, the steps are often meant to interact – if we imagine that monadic values are computations, all the steps should be composed (monadically speaking, multiplied out) into one, big computation; if we view monads as algebraic theories, we should take into account that operations in one layer have their arguments in the next layer. Obviously, not every monad is coinductive in this sense, because the notion of multiplication of infinitely many layers is not always well-defined. Thus, to capture the notion of coinduction in the monadic context, we adopt a property called *complete iterativity*, introduced by Elgot *et al.* [16] and later studied by Adámek *et al.* [4,30]. A monad M is *completely iterative* (is a 'cim') if it is equipped with an additional coinductive structure: certain ('guarded') morphisms $e : X \to M(X + A)$, where X represents variables (seeds of the corecursion) and A is an object of parameters (final values), have unique solutions $e^\dagger : X \to MA$ coherent with the monadic structure of M.

Not too surprisingly, the usual notion of coinduction is captured by

free completely iterative monads (informally: layers do not interact). The free completely iterative monad generated by an endofunctor F is given as $F^\infty A = \nu X.FX + A$ with the monadic structure given by substitution in A. An ordinary coalgebra $X \to FX$ can be seen as a guarded morphism $X \to F^\infty(X + 0)$, where 0 is the initial object of the base category, with the unique solution $X \to F^\infty 0 \cong \nu F$.

1.3 Contributions

The monad $TA = \nu X.M(\Sigma X + A)$ can alternatively be given as $M(\Sigma M)^\infty$. In Section 3, we generalise this construction to MS^∞, where S is any right module of M (all the necessary definitions and notations are given in Section 2). We give it a monadic structure and prove that it is completely iterative. Moreover, if M is also a cim, MS^∞ is a cim both 'vertically' (informally, we build new levels of the free structure) and 'horizontally' (we unfold more M structure) simultaneously.

In Section 4, we turn our attention back to the instance $M(\Sigma M)^\infty$. We show that it enjoys a certain universal property, which entails that it is the coproduct of Σ^∞ and M in the category of cims. In Section 5, we discuss corollaries and potential applications of our construction in semantics and programming.

We present only short outlines of some proofs. For the full proofs, consult the associated technical appendix available online at http://www.cs.ox.ac.uk/people/maciej.pirog/crm-appendix.pdf.

2 Idealised and completely iterative monads

In the rest of the paper, we work in a base category $\mathbb{B}$ with finite coproducts. For brevity, we assume strict associativity of the coproduct bifunctor. The left and right injections are called inl and inr respectively. For an endofunctor $F : \mathbb{B} \to \mathbb{B}$, we denote the carrier of the initial F-algebra as μF, and the carrier the final F-coalgebra as νF. The unique morphism from a coalgebra $\langle A, g : A \to FA\rangle$ to the final coalgebra $\langle \nu F, \xi : \nu F \to F\nu F\rangle$ is written as $(\!|g|\!)$. We use the letters M, N, T for monads. Their monadic structure is always denoted as η (unit) and μ (multiplication), possibly with superscripts to assign the natural transformations to the appropriate monad. The category of monads and monad morphisms is denoted as Mnd, while the category of Eilenberg-Moore algebras of a monad M is denoted as M-Malg.

Definition 2.1 *Let M be a monad. An endofunctor $\overline{M}$ together with a natural transformation (an* action*) $\overline{\mu} : \overline{M}M \to \overline{M}$ is called a* (right) M-module *if $\overline{\mu} \cdot \overline{M}\eta = \mathsf{id} : \overline{M} \to \overline{M}$ and $\overline{\mu} \cdot \overline{\mu}M = \overline{\mu} \cdot \overline{M}\mu : \overline{M}M^2 \to \overline{M}$. We define a morphism between two M-modules $\langle \overline{M}, \overline{\mu}\rangle$ and $\langle \widetilde{M}, \widetilde{\mu}\rangle$ as a natural transformation*

$f : \overline{M} \to \widetilde{M}$ *such that* $\widetilde{\mu} \cdot fM = f \cdot \overline{\mu} : \overline{M}M \to \widetilde{M}$.

Slightly abusing notation, we may denote a module $\langle \overline{M}, \overline{\mu} \rangle$ simply as $\overline{M}$.

Example 2.2 The following are examples of modules:

(i) For a monad M, the pair $\langle M, \mu^M \rangle$ is an M-module.

(ii) Let $m : M \to T$ be a monad morphism. Then the pair $\langle T,\ \mu^T \cdot Tm \rangle$ is an M-module.

(iii) Let $\langle \overline{M}, \overline{\mu}^M \rangle$ be an M-module and F be an endofunctor. Then, $\langle F\overline{M}, F\overline{\mu}^M \rangle$ is an M-module.

(iv) With the definitions as above, let $\lambda : TM \to MT$ be a distributive law between monads. The pair $\langle \overline{M}T,\ (\overline{\mu}^M * \mu^T) \cdot \overline{M}\lambda T \rangle$ is a module of the induced monad MT.

(v) If $\langle \overline{M}, \overline{\mu}^M \rangle$ and $\langle \widetilde{M}, \widetilde{\mu}^M \rangle$ are two M-modules, the pair $\langle \overline{M} + \widetilde{M},\ \overline{\mu}^M + \widetilde{\mu}^M \rangle$ is also an M-module.

(vi) Let F be an endofunctor with a right adjoint U. Then, F is an UF-module with the action given by $\varepsilon F : FUF \to F$, where ε is the counit of the adjunction.

Definition 2.3 *An* idealised monad *is a triple consisting of a monad M, an M-module $\langle \overline{M}, \overline{\mu}^M \rangle$, and a module homomorphism $\sigma : \langle \overline{M}, \overline{\mu}^M \rangle \to \langle M, \mu^M \rangle$. We say that M is* idealised *with $\langle \overline{M}, \overline{\mu}^M \rangle$. If $M = \overline{M} + \mathsf{Id}$, we say that M is* ideal. *For an endofunctor F, a natural transformation $k : F \to M$ is* ideal *if it factors through σ.*

If not stated otherwise, for an idealised monad M, by $\overline{\mu}^M$ we always denote the action of the associated module $\overline{M}$, and by σ^M the associated module homomorphism.

Example 2.4 Extending Example 2.2 (iv) and (v), it holds that:

(i) Let M be idealised with $\overline{M}$ and $\lambda : TM \to MT$ be a distributive law between monads. The induced monad MT is idealised with $\overline{M}T$. The associated module morphism is given by $\sigma^M T : \overline{M}T \to MT$.

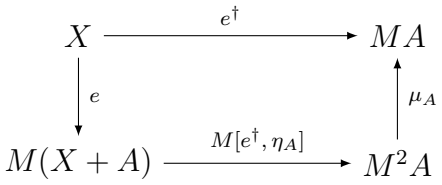
(ii) Let M be idealised with $\overline{M}$ as well as with $\widetilde{M}$. Then, M is idealised with $\overline{M} + \widetilde{M}$. The associated module morphism is given by $[\sigma, \sigma'] : \overline{M} + \widetilde{M} \to M$, where σ and σ' are the respective associated morphisms of the two modules.

Definition 2.5 *Let M be a monad idealised with $\overline{M}$. A morphism $e : X \to M(X + A)$ is called a* guarded equation morphism *if it factors as follows:*

$$X \dashrightarrow \overline{M}(X + A) + A \xrightarrow{[\sigma_{X+A},\ \eta_{X+A} \cdot \mathsf{inr}_{X,A}]} M(X + A)$$

We call a morphism $e^\dagger : X \to MA$ a solution *of e if the following diagram*

commutes:



An idealised monad M is completely iterative *(is a 'cim') if every guarded equation morphism has a unique solution.*

A monad morphism $m : M \to T$, where T is idealised with $\langle \overline{T}, \overline{\mu}^T \rangle$, is said to preserve solutions *if there exists a natural transformation $\overline{m} : \overline{M} \to \overline{T}$, such that $m \cdot \sigma^M = \sigma^T \cdot \overline{m} : \overline{M} \to T$.*

We denote the category of all cims and solution-preserving monad morphisms as Cim.

Note that we use a different notion of morphisms between cims than Adámek *et al.* [5], whose morphisms – called *idealised monad morphisms* – preserve also the structure of modules. The name 'solution-preserving' comes from the fact that for an equation morphism e and m as in the definition above, it holds that $m_{X+A} \cdot e$ is guarded and that $(m_{X+A} \cdot e)^\dagger = m_A \cdot e^\dagger$ (see the proof of a theorem by Milius [30, Prop. 5.9]).

An important example of a cim is the free cim Σ^∞ generated by an endofunctor Σ. Intuitively, it captures the monad of non-well-founded Σ-terms. Given an endofunctor Σ (a signature), if the final coalgebra $\langle \nu X.\Sigma X + A, \xi_A \rangle$ exists for all objects A, then we define $\Sigma^\infty A = \nu X.\Sigma X + A$. One can show that it is functorial in A, with the obvious action on morphisms, and that it has a monadic structure given by substitution in A, which we denote as η^∞ and μ^∞. The monad Σ^∞ is ideal and completely iterative. We define a natural transformation $\mathsf{emb} : \Sigma \to \Sigma^\infty$ as:

$$\mathsf{emb}_A = \left(\Sigma A \xrightarrow{\Sigma \eta^\infty} \Sigma\Sigma^\infty A \xrightarrow{\mathsf{inl}} \Sigma\Sigma^\infty A + A \xrightarrow{\xi^{-1}} \Sigma^\infty A \right)$$

As discussed by Aczel *et al.* [4], Σ^∞ is the free cim generated by Σ. Intuitively, this means that every ideal interpretation of Σ in a cim M extends in a unique way to an interpretation of the entire structure in M. Formally:

Theorem 2.6 *For a cim M and an ideal natural transformation $k : \Sigma \to M$, there exists a unique monad morphism $\iota(k) : \Sigma^\infty \to M$ such that $k = \iota(k) \cdot \mathsf{emb}$. Moreover, the morphism $\iota(k)$ preserves solutions.*

We also need the following cancellation property:

Lemma 2.7 *For a cim M and a solution-preserving monad morphism $m : \Sigma^\infty \to M$, the composition $m \cdot \mathsf{emb}$ is an ideal natural transformation, and $\iota(m \cdot \mathsf{emb}) = m$.*

3 Monadic structure and complete iterativity

Let $\langle S, \overline{\mu}^S \rangle$ be a right M-module such that S^∞ exists. We give a monadic structure to MS^∞ via a distributive law [11]. This construction is an adaptation of Hyland, Plotkin, and Power's proof [26] that the inductive resumptions $M(\Sigma M)^*$ form a monad. We use the fact due to Adámek, Milius, and Velebil [7] that the category of complete Elgot algebras is strictly monadic over the base category $\mathbb{B}$. Note that we cannot employ Uustalu's construction on parametrised monads [41] (successfully used by Goncharov and Schröder [21] in the special case of MM^∞), since MS^∞ is not in general given by the carrier of a final coalgebra; moreover, we make extensive use of the distributive law later in the paper. We start by recalling the definition of Elgot algebras [7].

Definition 3.1 *Let H be an endofunctor. For two objects A and X, we call a morphism $e : X \to HX + A$ a* flat equation morphism. *We call a morphism $e^\dagger : X \to A$ a* solution *in an H-algebra $a : HA \to A$ if the following diagram commutes:*

$$\begin{array}{ccc} X & \xrightarrow{\quad e^\dagger \quad} & A \\ {\scriptstyle e}\downarrow & & \uparrow{\scriptstyle [a,\mathsf{id}]} \\ HX + A & \xrightarrow{\quad He^\dagger + \mathsf{id} \quad} & HA + A \end{array}$$

Just like in the case of cims, we denote the solutions in Elgot algebras by $(\text{-})^\dagger$ or $(\text{-})^\ddagger$. This overloading should not impose any confusion, since we are always clear about the types.

Definition 3.2 *For flat equation morphisms $e : X \to HX + Y$ and $f : Y \to HY + A$, and a morphism $h : Y \to Z$, we define two operations. The first one, $\triangleleft$, 'renames' the parameter Y using the morphism h:*

$$h \triangleleft e = \left(X \xrightarrow{e} HX + Y \xrightarrow{\mathsf{id}+h} HX + Z \right)$$

The second one, $\oplus$, unfolds the flat equation morphisms 'in parallel':

$$f \oplus e = \left(X + Y \xrightarrow{[e,\mathsf{inr}]} HX + Y \xrightarrow{\mathsf{id}+f} HX + HY + A \xrightarrow{[H\mathsf{inl},H\mathsf{inr}]+\mathsf{id}} H(X+Y) + A \right)$$

Definition 3.3 *For an endofunctor H, a* complete Elgot H-algebra *is a triple $\langle A, a : HA \to A, (\text{-})^\dagger \rangle$, where $(\text{-})^\dagger$ assigns to every flat equation morphism $e : X \to HX + A$ a solution $e^\dagger : X \to A$ such that the following two conditions hold:*

- *(Functoriality) For two equation morphisms $e : X \to HX + A$ and $f : Y \to HY + A$ understood as $H(\text{-}) + A$ coalgebras, let $h : X \to Y$ be a coalgebra*

homomorphism, that is $f \cdot h = (Hh + \mathsf{id}_A) \cdot e$. *Then,* $e^\dagger = f^\dagger \cdot h$.

- *(Compositionality) For two equation morphisms* $e : X \to HX + Y$ *and* $f : Y \to HY + A$ *it holds that* $(f^\dagger \triangleleft e)^\dagger = (f \mathbin{⦶} e)^\dagger \cdot \mathsf{inl}$.

Definition 3.4 *For two complete Elgot* H*-algebras* $\langle A, a, (\text{-})^\dagger \rangle$ *and* $\langle B, b, (\text{-})^\ddagger \rangle$, *a morphism* $h : A \to B$ *is said to* preserve solutions *if for every flat equation morphism* $e : X \to HX + A$ *it holds that* $(h \triangleleft e)^\ddagger = h \cdot e^\dagger$. *Complete Elgot* H*-algebras and solution-preserving morphisms form a category, which we denote as* H-Elgot.

Theorem 3.5 (Adámek, Milius, Velebil [7]) *The obvious forgetful functor* $U_{\text{Elg}} : H\text{-Elgot} \to \mathbb{B}$ *is strictly* H^∞*-monadic (or simply 'monadic' in Mac Lane's terminology [29, Ch. 6]), and hence* H-Elgot $\cong H^\infty$-Malg.

Construction 3.6 *Recall that* $\langle S, \overline{\mu}^S \rangle$ *is a right* M*-module. For a complete Elgot algebra* $\langle A, a : SA \to A, (\text{-})^\dagger \rangle$ *we define an algebra* $\langle MA,\, a' : SMA \to MA,\, (\text{-})^\ddagger \rangle$ *as follows:*

$$a' = \left(SMA \xrightarrow{\overline{\mu}^S} SA \xrightarrow{a} A \xrightarrow{\eta^M} MA \right)$$

Let $e : X \to SX + MA$ *be a flat equation morphism. We define an auxiliary morphism* $|e|$ *and a solution* $e^\ddagger$*:*

$$|e| = \left(SX + A \xrightarrow{Se + \mathsf{id}} S(SX + MA) + A \xrightarrow{\mathsf{flat} + \mathsf{id}} S(SX + A) + A \right)$$

$$e^\ddagger = \left(X \xrightarrow{e} SX + MA \xrightarrow{\mathsf{inl} + \mathsf{id}} SX + A + MA \xrightarrow{|e|^\dagger + \mathsf{id}} A + MA \xrightarrow{[\eta^M, \mathsf{id}]} MA \right)$$

where $\mathsf{flat}_{A,B}$ *is given as:*

$$S(A + MB) \xrightarrow{S(\eta^M + \mathsf{id})} S(MA + MB) \xrightarrow{S[M\mathsf{inl}, M\mathsf{inr}]} SM(A + B) \xrightarrow{\overline{\mu}^S} S(A + B)$$

Lemma 3.7 *The triple* $\langle MA, a', (\text{-})^\ddagger \rangle$ *from Construction 3.6 is a complete Elgot algebra. Moreover, the assignment* $\langle A, a, (\text{-})^\dagger \rangle \mapsto \langle MA, a', (\text{-})^\ddagger \rangle$ *on objects and* $f \mapsto Mf$ *on morphisms is an endofunctor on* S-Elgot *with a monadic structure given by the monadic structure of* M.

Theorem 3.8 *The composition* MS^∞ *is a monad.*

Proof. The assignment from Lemma 3.7 is a monad, so it is a lifting of M to S-Elgot with respect to U_{Elg}. Thus, by Theorem 3.5, it is a lifting of M to S^∞-Malg. This induces a distributive law between monads $\lambda : S^\infty M \to MS^\infty$, which gives a monadic structure to MS^∞. □

Example 3.9 Let $\mathbb{B}$ be Set, $\mathcal{D}$ be the monad of discrete probability distributions, $O = \{\mathsf{a}, \mathsf{b}, \ldots\}$ be a set, and $\Sigma X = O \times X$ be an endofunctor. An element of the carrier of the monad of the monad $\mathcal{D}(\Sigma\mathcal{D})^\infty X$ is a countably

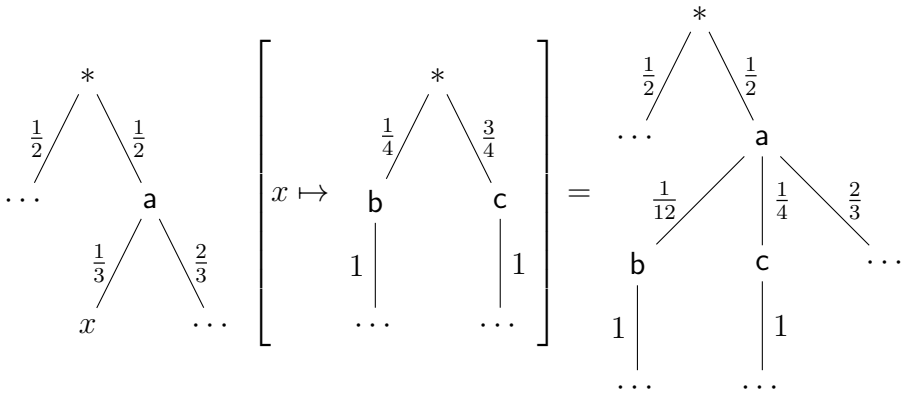


Fig. 1. Example of a substitution in the $\mathcal{D}(O \times \mathcal{D}(\text{-}))^\infty$ monad.

branching, possibly infinite decision tree in which the nodes (except for the root) are labelled with elements of the set O, edges with probabilities, and leaves with elements of X. Such a structure can be understood as a denotation of a possibly non-terminating, probabilistic process that produces a stream of elements of O as its output.

From the technical perspective, it is important that the root has no label – we take a probabilistic step before generating a label and a probabilistic step before reaching a leaf. This way, when we substitute a tree for a variable, we take two probabilistic steps before generating a label or reaching a leaf. The monadic structure of $\mathcal{D}(\Sigma\mathcal{D})^\infty X$ takes care of flattening these to one probabilistic step by multiplying out the adjacent distributions; see Figure 1 for an example.

Example 3.10 In a cartesian closed category, we can define a version of the state monad that keeps track of all (possibly infinitely many) intermediate states. It is similar but not identical to 'states' given in [39] (compare also Ahman and Uustalu's update monads [8]). Fix an object of states A and consider the reader monad $\mathcal{R}X = X^A$. The writer $\mathcal{W}X = X \times A$ is an $\mathcal{R}$-module. The action can be given as $\langle \mathsf{ev}^{\mathsf{A}}_{\mathsf{X}}, \mathsf{outr} \rangle : \mathcal{W}\mathcal{R}X = X^A \times A \to X \times A = \mathcal{W}X$, where ev is the evaluation morphism of the exponential object, outr is the right projection, and $\langle \text{-}, \text{-} \rangle$ is the product mediator. Intuitively, for an initial state, the monad $\mathcal{R}\mathcal{W}^\infty = ((\text{-} \times A)^\infty)^A$ produces a (possibly infinite) stream of intermediate states $\mathcal{W}^\infty$. If the stream is finite, it is terminated with a final value of the computation. The monad $\mathcal{R}\mathcal{W}^\infty$ is an instance of the resumption monad that in general is not given by a final coalgebra.

Now, assume that M is completely iterative with respect to a module $\overline{M}$. Note that an 'ordinary' monad is always a cim with respect to the module C_0 (the constant functor returning the initial object), so this assumption does not

narrow down the choice of M. We prove MS^∞ to be a cim with respect to the module $\overline{M}S^\infty + MSS^\infty$. This means that each coinductive step can unfold the structure 'horizontally' (by unfolding more structure of M), 'vertically' (unfolding the free structure), or both.

Definition 3.11 *For a monad T, a* distributive law *of a T-module $\overline{T}$ over a monad M is a distributive law between monads $\lambda : TM \to MT$ together with a natural transformation $\overline{\lambda} : \overline{T}M \to M\overline{T}$ such that the obvious analogues of the diagrams for distributive laws between monads commute (except for the diagram for η^T, since in general $\overline{T}$ is not a monad). Moreover, if T is idealised with $\overline{T}$, we say that the tuple $\langle \lambda, \overline{\lambda} \rangle$* preserves modules *if $M\sigma^T \cdot \overline{\lambda} = \lambda \cdot \sigma^T M : \overline{T}M \to MT$.*

Lemma 3.12 *Let T be an idealised monad and let $\langle \lambda, \overline{\lambda} \rangle$ be a module-preserving distributive law of $\overline{T}$ over M. Then the induced monad MT is idealised with $M\overline{T}$.*

One can show that $\lambda : S^\infty M \to MS^\infty$ from the proof of Theorem 3.8 can be extended to a module-preserving distributive law of SS^∞ over M. Hence, Lemma 3.12 together with Example 2.4 lead us to the following corollary:

Corollary 3.13 *The monad MS^∞ is idealised with respect to $\overline{M}S^\infty + MSS^\infty$.*

We describe a solution in MS^∞ as a two-step process. Intuitively, given an equation morphism $e : X \to MS^\infty(X + A)$, we first unfold 'horizontally' via the completely iterative structure of M. We are left with what can be seen as an equation morphism in a monad induced by the distributive law λ in the Kleisli category of M. This morphism (the 'vertical' unfolding) has a unique solution, too, which is the desired solution in MS^∞. First, we need the following technical lemma:

Lemma 3.14 *Let M be a cim. Let $e : X \to M(X + A + B)$ factor as follows:*

$$X \dashrightarrow \overline{M}(X + A + B) + MA + B \xrightarrow{[\sigma,\ M(\mathsf{inl}\cdot\mathsf{inr}),\ \eta\cdot\mathsf{inr}]} M(X + A + B)$$

The morphism e has a unique solution $e^\dagger : X \to M(A + B)$.

The monad NS^∞ has the following property, which is stronger than being a cim:

Lemma 3.15 *Let $e : X \to MS^\infty(X + A)$ be a morphism that factors as follows:*

$$X \dashrightarrow M(SS^\infty(X + A) + A) \xrightarrow{M[\sigma^\infty,\ \eta^\infty\cdot\mathsf{inr}]} MS^\infty(X + A)$$

Then, e has a unique solution $e^\ddagger$ in MS^∞.

Proof. Consider the Kleisli category of M, denoted as $\mathcal{K}\ell(M)$. For a morphism $f : A \to MB$ in $\mathcal{K}\ell(M)$, we define a $\mathbb{B}$-morphism $\widehat{f} = \overline{\mu}^S_B \cdot Sf : SA \to SB$. We define an endofunctor G on $\mathcal{K}\ell(M)$ given as $GA = SA$ on objects and $G(f : A \to MB) = \eta^M_B \cdot S\widehat{f} : SA \to MSB$ on morphisms. The distributive law λ induces a monad $\langle\!\langle S^\infty \rangle\!\rangle$ on $\mathcal{K}\ell(M)$ given by $\langle\!\langle S^\infty \rangle\!\rangle A = S^\infty A$ and $\langle\!\langle S^\infty \rangle\!\rangle (f : A \to MB) = \lambda_B \cdot S^\infty f : S^\infty A \to MS^\infty B$. One can show that $\langle\!\langle S^\infty \rangle\!\rangle$ is the free cim generated by G in $\mathcal{K}\ell(M)$. The morphism e is a guarded equation morphism in $\langle\!\langle S^\infty \rangle\!\rangle$, so it has a unique solution $e^\ddagger : X \to M\langle\!\langle S^\infty \rangle\!\rangle A$. One can verify that it is the desired morphism and that it is unique. □

Now, we can prove the main theorem:

Theorem 3.16 *The monad MS^∞ is completely iterative with respect to the module $\overline{M}S^\infty + MSS^\infty$.*

Proof. In this proof, for brevity, we denote S^∞ as T and its module SS^∞ as $\overline{T}$. Let $e : X \to MT(X + A)$ be a guarded equation morphism. This means that it factors as $X \dashrightarrow \overline{M}T(X+A) + M\overline{T}(X+A) + A \xrightarrow{[\sigma^M, M\sigma^T, \eta \cdot \mathsf{inr}]} MT(X+A)$. Since T is an ideal monad, there exist isomorphisms $T(X + A) \cong \overline{T}(X + A) + X + A \cong X + \overline{T}(X+A) + A$. Thus, e factors as $X \dashrightarrow \overline{M}(X + \overline{T}(X+A) + A) + M\overline{T}(X + A) + A \xrightarrow{[\sigma^M, M(\mathsf{inl} \cdot \mathsf{inr}), \eta \cdot \mathsf{inr}]} M(X + \overline{T}(X + A) + A)$, and it is a guarded equation morphism in the monad M in the sense of Lemma 3.14. Therefore, there exists a unique solution $e^\dagger : X \to M(\overline{T}(X + A) + A)$. The morphism $M[\sigma^T, \eta^T \cdot \mathsf{inr}] \cdot e^\dagger : X \to MT(X + A)$ is a guarded equation morphism in the sense of Lemma 3.15, so it has a unique solution $(M[\sigma^T, \eta^T \cdot \mathsf{inr}] \cdot e^\dagger)^\ddagger : X \to MTA$. One can verify that it is a unique solution of e in MT. □

Example 3.17 Consider the monad $\mathcal{D}(\Sigma\mathcal{D})^\infty$ from Example 3.9. It is a cim, which gives us a semantics for probabilistic processes with sequential composition. An equation morphism $e : X \to \mathcal{D}(\Sigma\mathcal{D})^\infty(X + A)$ can be understood as a transition system, where X is the state space. The solution $e^\dagger : X \to \mathcal{D}(\Sigma\mathcal{D})^\infty A$ is a denotational semantics: for an initial state, it gives us the decision tree, while all the intermediate states are forgotten (they are *internal* to the computation). The solution diagram amounts to adequacy.

For an example of horizontal and vertical complete iterativity, consider the monad $\mathcal{D}'X = \mathcal{D}(1 + X)$ for the terminal object 1, which denotes failure. It is a cim with respect to the module that consists of those values of $\mathcal{D}'$ that fail with the probability at least 0.5. Each transition of a process denoted by $\mathcal{D}'(\Sigma\mathcal{D}')^\infty$ can either produce an output from the set O or perform a silent step, but with the probability of failure at least 0.5.

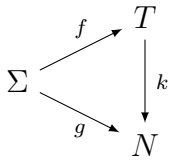
4 Universal property and coproduct

A particularly interesting instance of MS^∞ is $M(\Sigma M)^\infty$. In this section, we investigate its universal property: for a cim N, a monad morphism $M \to N$, and an ideal natural transformation $\Sigma \to N$, there exists a unique coherent monad morphism $M(\Sigma M)^\infty \to N$. Informally, morphisms that 'interpret' every level of the structure of a resumption in another cim uniquely extend to an interpretation of the whole structure. First, we define three 'injections':

$$\begin{aligned}
\mathsf{liftl} &= \left(\Sigma^\infty \xrightarrow{\iota(\mathsf{emb}\cdot\Sigma\eta^M)} (\Sigma M)^\infty \xrightarrow{\eta^M(\Sigma M)^\infty} M(\Sigma M)^\infty\right) \\
\mathsf{liftr} &= \left(M \xrightarrow{M\eta^\infty} M(\Sigma M)^\infty\right) \\
\mathsf{liftf} &= \left(\Sigma \xrightarrow{\mathsf{emb}} \Sigma^\infty \xrightarrow{\mathsf{liftl}} M(\Sigma M)^\infty\right)
\end{aligned}$$

It is easy to see that liftl and liftr are monad morphisms (liftl is a composition of two monad morphisms). They are also solution-preserving, the former via $M\Sigma M(\Sigma M)^\infty$, the latter via $\overline{M}(\Sigma M)^\infty$.

Definition 4.1 *By Σ/Cim we denote the following category: objects are ideal natural transformations $(\Sigma \xrightarrow{f} T)$, where T is a cim; morphisms are (not necessarily solution-preserving) monad morphisms $k : T \to N$ such that the following diagram commutes:*



We define a forgetful functor $U : \Sigma/\textsc{Cim} \to \textsc{Mnd}$ as $U(\Sigma \xrightarrow{f} M) = M$ on objects and $Uk = k$ on morphisms.

Definition 4.2 *A monad M is called Σ-resumable if $(\Sigma M)^\infty$ exists. By Σ-Res we denote the full subcategory of Mnd with Σ-resumable cims as objects. We denote the inclusion functor by $I : \Sigma\textsc{-Res} \to \textsc{Mnd}$.*

We establish the universal property in terms of an I-relative adjunction. For a discussion on relative adjoints, see, for example, Altenkirch *et al.* [9].

Theorem 4.3 *The functor U has an I-relative left adjoint $F : \Sigma\textsc{-Res} \to \Sigma/\textsc{Cim}$ given by $FM = (\Sigma \xrightarrow{\mathsf{liftf}} M(\Sigma M)^\infty)$ on objects and $Fm = m * [\![(\Sigma m(\Sigma M)^\infty + \mathsf{id}) \cdot \xi]\!] = m * \iota(\mathsf{emb} \cdot \Sigma m)$ on morphisms.*

Proof. One can show that the morphism Fm is a monad morphism, so F is indeed a functor. Let M be a Σ-resumable monad, N be a cim, and $g : \Sigma \to N$

be an ideal natural transformation. We define the isomorphism

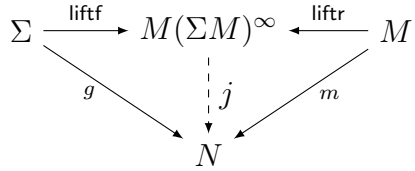
$$\lfloor\text{-}\rfloor : \Sigma/\textsc{Cim}[FM, (\Sigma \xrightarrow{g} N)] \cong \textsc{Mnd}[IM, U(\Sigma \xrightarrow{g} N)] : \lceil\text{-}\rceil$$

by

$$\begin{array}{ll} k : (\Sigma \xrightarrow{\mathsf{liftf}} M(\Sigma M)^\infty) \to (\Sigma \xrightarrow{f} N) & m : M \to U(\Sigma \xrightarrow{f} N) \\ \lfloor k \rfloor : M \to U(\Sigma \xrightarrow{f} N) & \lceil m \rceil : (\Sigma \xrightarrow{\mathsf{liftf}} M(\Sigma M)^\infty) \to (\Sigma \xrightarrow{f} N) \\ \lfloor k \rfloor = k \cdot M\eta^{(\Sigma M)^\infty} & \lceil m \rceil = \mu^N \cdot (m * \iota(\mu^N \cdot (f * m))) \end{array}$$

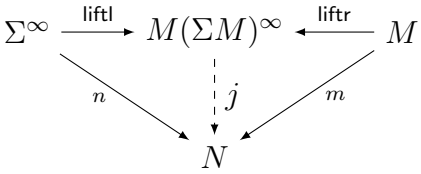
For $\lceil m \rceil$ to be a well-defined morphism in $\Sigma/\textsc{Cim}$, we note that $\lceil m \rceil \cdot \mathsf{liftf} = f$. Using the properties of $M(\Sigma M)^\infty$ and free cims, one can verify that $\lfloor\text{-}\rfloor$ is natural in M and g, and that $\lfloor\text{-}\rfloor$ and $\lceil\text{-}\rceil$ are mutual inverses. □

An alternative reading of Theorem 4.3 is that $(\Sigma \xrightarrow{\mathsf{liftf}} M(\Sigma M)^\infty)$ is the free object in $\Sigma/\textsc{Cim}$ generated by a Σ-resumable cim M. This means that for a cim N, an ideal natural transformation $g : \Sigma \to N$, and a monad morphism $m : M \to N$, there exists a unique monad morphism $j : M(\Sigma M)^\infty \to N$ such that the following diagram commutes (one can easily see that liftr is the unit of the relative adjunction):



Note that if $M = \mathsf{Id}$, the diagram above instantiates to the fact that Σ^∞ is indeed the free cim generated by Σ. More precisely, the identity monad Id is initial in $\textsc{Mnd}$ (the unique monad morphism $! : \mathsf{Id} \to N$ is given by the unit of N), so the right-hand side of the diagram above becomes trivial, and what is left is exactly the diagram from Theorem 2.6.

Also, for a solution-preserving monad morphism $n : \Sigma^\infty \to N$, the composition $n \cdot \mathsf{emb} : \Sigma \to N$ is an ideal natural transformation. For a solution-preserving monad morphism $m : M \to N$, there exists a unique monad morphism $j : M(\Sigma M)^\infty \to N$ such that $m = j \cdot \mathsf{liftr}$ and $n \cdot \mathsf{emb} = j \cdot \mathsf{liftf} = j \cdot \mathsf{liftl} \cdot \mathsf{emb}$. This means that $\iota(n \cdot \mathsf{emb}) = \iota(j \cdot \mathsf{liftl} \cdot \mathsf{emb})$, hence, by Lemma 2.7, we get $n = j \cdot \mathsf{liftl}$. Therefore, the morphism j uniquely makes the following diagram commute:



(1)

The morphism $j = U\lceil m \rceil_{M,n\cdot\mathsf{emb}}$ does not necessarily preserve solutions, even though liftl, liftr, m, and n do. Informally, the action of j on the right component of $M(\Sigma M)^\infty$'s module $\overline{M}(\Sigma M)^\infty + M\Sigma M(\Sigma M)^\infty$ is not guaranteed to take the leading M into $\overline{N}$. Nevertheless, we can amend the situation if we know that N is a cim with respect to a two-sided module:

Definition 4.4 *A two-sided module of a monad N is its right module $\langle \overline{N}, \overline{\mu}^{\mathsf{R}} \rangle$ together with a natural transformation $\overline{\mu}^{\mathsf{L}} : N\overline{N} \to \overline{N}$ such that the obvious diagrams mirroring those of Definition 2.1 commute and $\overline{\mu}^{\mathsf{R}} \cdot \overline{\mu}^{\mathsf{L}} N = \overline{\mu}^{\mathsf{L}} \cdot N\overline{\mu}^{\mathsf{R}} : N\overline{N}N \to \overline{N}$. Similarly, we adjust the definition of homomorphisms between modules and idealised monads. We denote the category of monads that are completely iterative with respect to two-sided ideals and solution-preserving monad morphisms as* TCim.

In the context of the properties studied in this paper, we can switch from Cim to TCim at no cost:

Theorem 4.5 *The category* TCim *is a full reflective subcategory of* Cim. *In other words, the obvious embedding functor $U_{\mathrm{TC}} :$ TCim $\to$ Cim has a left adjoint F_{TC}.*

In practise, this means that for a monad N completely iterative with respect to a right ideal $\overline{N}$, there exists a two-sided ideal $\widetilde{N}$ (given by $N\overline{N}$) and a module homomorphism $\overline{N} \to \widetilde{N}$ (that is, every equation morphism guarded via $\overline{N}$ is also guarded via $\widetilde{N}$), such that N is completely iterative with respect to $\widetilde{N}$. In such a case, since j from the diagram (1) is equal to $\mu^N \cdot (m * \iota(\mu^N \cdot ((n \cdot \mathsf{emb}) * m)))$ and the right-hand side argument of the left-most $*$ preserves solutions (Theorem 2.6), the morphism j is solution-preserving too, which can be verified by a simple diagram chase. In general, the module of Σ^∞ is two-sided, but the module of $M(\Sigma M)^\infty$ is not. Thus, taking the reflection of the diagram (1) in TCim, we obtain the following corollary, which is a 'completely iterative' counterpart of Hyland, Plotkin and Power's result that $M(\Sigma M)^*$ is a coproduct of Σ^* and M in Mnd [26]:

Corollary 4.6 *For an endofunctor Σ and a monad M completely iterative with respect to a two-sided ideal, the F_{TC}-image of $M(\Sigma M)^\infty$ (that is, $M(\Sigma M)^\infty$ idealised with $M(\Sigma M)^\infty(\overline{M}(\Sigma M)^\infty + M\Sigma M(\Sigma M)^\infty)$) is the coproduct of Σ^∞ and M in* TCim.

5 Discussion

5.1 Complete iterativity

The results presented in this paper are in general contributions to the study of completely iterative monads [4]. We give new examples of cims and describe coproducts with free cims. Note that if M is ideal, then so is MS^∞, which

means that our constructions scale to the category of ideal cims, if one prefers to work in such a setting.

Our results suggest a form of 'least- vs greatest fixed points' duality between ordinary monads and cims: free objects are given by $F^*A = \mu X.FX + A$ and $F^\infty A = \nu X.FX + A$ respectively, and coproducts with free objects by $M(\Sigma M)^*$ and $M(\Sigma M)^\infty$. There are other constructions on monads that involve initial algebras, for example the coproduct of two ideal monads, as shown by Ghani and Uustalu [20]. We conjecture that a similar construction with ν in place of μ yields the coproduct in the category of ideal cims.

Adámek *et al.* [5,6] consider also a finitary case: they define an *iterative monad* (without 'completely') as a finitary monad on a locally finitely presentable category, such that all guarded equation morphisms with finitely presentable object of variables have unique solutions. Similarly, there exists a finitary version of complete Elgot algebras (namely, *Elgot algebras*) together with an analogue of Theorem 3.5. This suggests that the presented constructions should scale to the finitary case, but we have not yet worked out the details.

Elgot's original results were set in the context of algebraic theories rather than general category theory. As a future direction of research, it would be interesting to look at structures based on resumptions from the point of view of algebraic specification (operations and equations), especially those that could be used in semantics and programming, like the logging monad from Example 3.10.

5.2 Semantics and programming

As suggested by the present authors in a previous paper [39], models in the style of Moggi [34] of effects generating observable behaviour (such as I/O) require a form of complete iterativity, given that programs need not terminate. Thus, by understanding the category of cims, one hopes for a better understanding of such effects. For example, if one pursues modularity, the coproduct in Mnd of two cims is not in general completely iterative. So, a much better notion of the 'smallest' amalgam of such effects would be their coproduct in TCim. This has a practical aspect: the Haskell programming language is equipped with a single 'all-inclusive' *IO* monad, incorporating effects as diverse as textual interaction, file handling, system calls, the foreign function interface, random number generation, and concurrency; one would hope for more fine-grained components that indicate the actual impure effects in use (see Peyton Jones [27] for discussion).

Resumptions-like structures are used as denotations of processes, that is programs that exhibit observable behaviour in the course of execution (see Abramsky [1]). A monadic structure captures the notion of composition, which

allows the resumption monad to serve as the basis of a programming calculus in the style of Moggi [34]. For example, Goncharov and Schröder [21] give a simple semantics for asynchronous side-effecting concurrent processes using the monad MM^∞. We hope that the richer structure of MS^∞ can be used to describe more advanced behaviours and synchronisation of processes along the lines of Abramsky's interaction categories [2].

In pure functional programming, monads are often used as an embedded domain-specific language (EDSL) for representing computational effects, built from atomic actions using functoriality and monadic structure. Very often such monadic values describe not necessarily terminating (thus, in a sense, coinductive) programs with non-trivial, parallel resource management, like lazy I/O; see, for example, Kiselyov's iteratees [28]. Such programs are often represented by data structures similar to monadic resumptions, which are in a close relationship with the outer, 'imperative' monad, such as Haskell's *IO*. The relationship between the two can be formalised by the operations discussed in this paper: *IO* actions can be lifted to the level of resumptions (liftr), while the whole EDSL program can be executed, that is flattened back into *IO*, which can be modelled by the universal property.

5.3 Related work

The notion of complete iterativity was introduced by Elgot [16], and later studied by Aczel *et al.* [4,30]. The properties of the monad Σ^∞ were studied also by Moss [35] and Ghani *et al.* [19]. Milius and Moss used Elgot algebras [7] to describe solutions to recursive program schemes [31]. The composition $M(\Sigma M)^\infty$ has been previously known to be a monad and a cim in the vertical manner (that is, with respect to the module $M\Sigma M(\Sigma M)^\infty$) [21,39]. In contrast to those results, our proofs do not depend on the resumption monad being given by a final coalgebra.

Resumptions were used in semantics of concurrency in many different shapes and under different names. A domain-theoretic approach to resumptions was taken by Milner [32], Plotkin [40], and Papaspyrou [37]. Interaction trees, that is final coalgebras of functors of the shape $\mathcal{P}((\text{-}) \times O)^I$ on Set, where $\mathcal{P}$ is the powerset functor, were extensively used in Abramsky's interaction categories [2] (the existence of such final coalgebras was assured by employing Aczel's non-well-founded set theory [3]).

In programming, different forms of resumptions have been independently rediscovered dozens of times, and used for flexible structuring of programs, though usually without much formal treatment. In the Lisp community, resumptions were dubbed 'engines' (Haynes and Friedman [25], Dybvig and Hieb [15]) or 'trampolined' programs (Ganz, Friedman, and Wand [18]), while in the Haskell libraries they can be found under the name 'free monad trans-

former' (since liftr from Section 4 is a monad morphism natural in M, the functor $M \mapsto M(\Sigma M)^\infty$ is a monad transformer in the sense of Moggi [33]). Different programming patterns that involve resumptions were discussed by Claessen [14], Kiselyov [28], Harrison [23], and the present authors [38]. Interleaving of data and effects in the algebraic context was also studied by Filinski and Støvring [17], and Atkey *et al.* [10].

In type theory, similar constructions were used to model interactive programs (Hancock and Setzer [22]), general recursion via guarded corecursion (Capretta [12]), or semantics of imperative languages (Nakata and Uustalu [36]).

Acknowledgements

We would like to thank Stefan Milius for his remarks on an early draft of this paper, and the anonymous reviewers for their detailed comments and helpful suggestions. The work reported here was partially supported by UK EPSRC-funded project *Reusability and Dependent Types* (EP/G034516/1).

References

[1] Samson Abramsky. Retracing some paths in process algebra. In Ugo Montanari and Vladimiro Sassone, editors, *International Conference on Concurrency Theory (CONCUR)*, volume 1119 of *Lecture Notes in Computer Science (LNCS)*, pages 1–17. Springer, 1996.

[2] Samson Abramsky, Simon Gay, and Rajagopal Nagarajan. Interaction categories and the foundations of typed concurrent programming. In *Proceedings of the 1994 Marktoberdorf Summer School on Deductive Program Design*, pages 35–113. Springer-Verlag, 1996.

[3] Peter Aczel. *Non-well-founded Sets*. Number 14 in Lecture Notes. Center for the Study of Language and Information, Stanford University, 1988.

[4] Peter Aczel, Jiří Adámek, Stefan Milius, and Jiří Velebil. Infinite trees and completely iterative theories: a coalgebraic view. *Theoretical Computer Science*, 300(1-3):1–45, 2003.

[5] Jiří Adámek, Stefan Milius, and Jiří Velebil. On rational monads and free iterative theories. *Electronic Notes in Theoretical Computer Science (ENTCS)*, 69:23–46, 2002.

[6] Jiří Adámek, Stefan Milius, and Jiří Velebil. Free iterative theories: A coalgebraic view. *Mathematical Structures in Computer Science*, 13(2):259–320, 2003.

[7] Jiří Adámek, Stefan Milius, and Jiří Velebil. Elgot algebras. *Logical Methods in Computer Science*, 2(5), 2006.

[8] Danel Ahman and Tarmo Uustalu. Update monads: Cointerpreting directed containers. In *Book of abstracts of the 19th Meeting of "Types for Proofs and Programs", TYPES 2013 (Toulouse, April 2013)*, pages 16–17, 2013.

[9] Thorsten Altenkirch, James Chapman, and Tarmo Uustalu. Monads need not be endofunctors. In *FOSSACS*, volume 6014 of *Lecture Notes in Computer Science (LNCS)*, pages 297–311. Springer, 2010.

[10] Robert Atkey, Neil Ghani, Bart Jacobs, and Patricia Johann. Fibrational induction meets effects. In *FOSSACS*, volume 7213 of *Lecture Notes in Computer Science (LNCS)*, pages 42–57. Springer, 2012.

[11] Jon Beck. Distributive laws. In *Seminar on Triples and Categorical Homology Theory*, volume 80 of *Lecture Notes in Mathematics*, pages 119–140. Springer Berlin / Heidelberg, 1969. 10.1007/BFb0083084.

[12] Venanzio Capretta. General recursion via coinductive types. *Logical Methods in Computer Science*, 1(2), 2005.

[13] Pietro Cenciarelli and Eugenio Moggi. A syntactic approach to modularity in denotational semantics. In *Category Theory and Computer Science*, Amsterdam, The Netherlands, 1993.

[14] Koen Claessen. A poor man's concurrency monad. *Journal of Functional Programming*, 9(3):313–323, 1999.

[15] R. Kent Dybvig and Robert Hieb. Engines from continuations. *Computer Languages*, 14(2):109–123, 1989.

[16] Calvin C. Elgot, Stephen L. Bloom, and Ralph Tindell. On the algebraic structure of rooted trees. *J. Comput. Syst. Sci.*, 16:362–399, 1978.

[17] Andrzej Filinski and Kristian Støvring. Inductive reasoning about effectful data types. In *International Conference on Functional Programming (ICFP)*, pages 97–110. ACM, 2007.

[18] Steven E. Ganz, Daniel P. Friedman, and Mitchell Wand. Trampolined style. In *International Conference on Functional Programming (ICFP)*, pages 18–27. ACM, 1999.

[19] Neil Ghani, Christoph Lüth, Federico De Marchi, and John Power. Algebras, coalgebras, monads and comonads. *Electronic Notes in Theoretical Computer Science (ENTCS)*, 44(1):128–145, 2001.

[20] Neil Ghani and Tarmo Uustalu. Coproducts of ideal monads. *Theoretical Informatics and Applications*, 38(4):321–342, 2004.

[21] Sergey Goncharov and Lutz Schröder. A coinductive calculus for asynchronous side-effecting processes. In *Symposium on Fundamentals of Computation Theory (FCT)*, volume 6914 of *Lecture Notes in Computer Science (LNCS)*, pages 276–287. Springer, 2011.

[22] Peter Hancock and Anton Setzer. Interactive programs in dependent type theory. In *Computer Science Logic (CSL)*, volume 1862 of *Lecture Notes in Computer Science (LNCS)*, pages 317–331. Springer, 2000.

[23] William L. Harrison. The essence of multitasking. In *Algebraic Methodology and Software Technology (AMAST)*, volume 4019 of *Lecture Notes in Computer Science (LNCS)*, pages 158–172. Springer, 2006.

[24] Ichiro Hasuo and Bart Jacobs. Traces for coalgebraic components. *Mathematical Structures in Computer Science*, 21(2):267–320, 2011.

[25] Christopher T. Haynes and Daniel P. Friedman. Engines build process abstractions. In *LISP and Functional Programming*, pages 18–24, 1984.

[26] Martin Hyland, Gordon D. Plotkin, and John Power. Combining effects: Sum and tensor. *Theoretical Computer Science*, 357(1-3):70–99, 2006.

[27] Simon Peyton Jones. Tackling the awkward squad: Monadic input/output, concurrency, exceptions, and foreign-language calls in Haskell. In *Engineering theories of software construction*, pages 47–96. IOS Press, 2002.

[28] Oleg Kiselyov. Iteratees. In *FLOPS*, volume 7294 of *Lecture Notes in Computer Science (LNCS)*, pages 166–181. Springer, 2012.

[29] Saunders Mac Lane. *Categories for the Working Mathematician.* Graduate Texts in Mathematics. Springer, 1998.

[30] Stefan Milius. Completely iterative algebras and completely iterative monads. *Information and Computation*, 196:1–41, 2005.

[31] Stefan Milius and Lawrence S. Moss. The category-theoretic solution of recursive program schemes. *Theoretical Computer Science*, 366(1-2):3–59, 2006.

[32] Robin Milner. Processes: A mathematical model of computing agents. In *Logic Colloquium '73*, Studies in logic and the foundations of mathematics, pages 157–173. North-Holland Pub. Co., 1975.

[33] Eugenio Moggi. An Abstract View of Programming Languages. Technical report, Edinburgh University, 1989.

[34] Eugenio Moggi. Notions of computation and monads. *Information and Computation*, 93(1):55–92, 1991.

[35] Lawrence S. Moss. Parametric corecursion. *Theoretical Computer Science*, 260(1-2):139–163, 2001.

[36] Keiko Nakata and Tarmo Uustalu. Resumptions, weak bisimilarity and big-step semantics for While with interactive I/O: An exercise in mixed induction-coinduction. In *SOS*, volume 32 of *Electronic Proceedings in Theoretical Computer Science (EPTCS)*, pages 57–75, 2010.

[37] Nikolaos S. Papaspyrou. A resumption monad transformer and its applications in the semantics of concurrency. In *Proceedings of the 3rd Panhellenic Logic Symposium, Anogia*, 2001.

[38] Maciej Piróg and Jeremy Gibbons. Tracing monadic computations and representing effects. volume 76 of *Electronic Proceedings in Theoretical Computer Science (EPTCS)*, pages 90–111, 2012.

[39] Maciej Piróg and Jeremy Gibbons. Monads for behaviour. *Electronic Notes in Theoretical Computer Science (ENTCS)*, 298:309–324, 2013. Mathematical Foundations of Programming Semantics (MFPS).

[40] Gordon D. Plotkin. A powerdomain construction. *SIAM: SIAM Journal on Computing*, 5(3):452–487, 1976.

[41] Tarmo Uustalu. Generalizing substitution. *RAIRO—Theoretical Informatics and Applications*, 37:315–336, 10 2003.