Algebraic and Coalgebraic Perspectives on Interaction Laws

Tarmo Uustalu[1,2] and Niels Voorneveld[2(✉)]

[1] Department of Computer Science, Reykjavik University, Reykjavik, Iceland
tarmo@ru.is
[2] Department of Software Science, Tallinn University of Technology, Tallinn, Estonia
niels.voorneveld@taltech.ee

Abstract. *Monad algebras*, turning computations over return values into values, are used to handle algebraic effects invoked by programs, whereas *comonad coalgebras*, turning initial states into environments ("cocomputations") over states, describe production of coalgebraic coeffects that can respond to effects. *(Monad-comonad) interaction laws* by Katsumata et al. describe interaction protocols between a computation and an environment. We show that any triple of those devices can be combined into a single algebra handling computations over state predicates. This method yields an isomorphism between the category of interaction laws, and the category of so-called *merge functors* which merge algebras and coalgebras to form combined algebras. In a similar vein, we can combine interaction laws with coalgebras only, retrieving Uustalu's stateful runners. If instead we combine interaction laws with algebras only, we get a novel concept of continuation-based runners that lift an environment of value predicates to a single predicate on computations of values. We use these notions to study different running examples of interactions of computations and environments.

Keywords: Monad algebras · Comonad coalgebras · Interaction laws · Runners · Monad morphisms · Effects · Coeffects

1 Introduction

Programs can exhibit effects which impact how they are run. Such effects (requests to the environment) may communicate with, invoke changes in, and otherwise influence the environment, producing coeffects (responses to the computation). How does one describe the protocols of such interactions?

Katsumata et al. [7] proposed to use *(monad-comonad) interaction laws*. We model the notion of effect using a monad T following Moggi [10], and the notion of coeffect using a comonad D, as pioneered by Power and Shkaravska [18]. The environment interacts with the effects, resolving some, ignoring others, and potentially producing new effects. A residual monad R is specified to capture these ignored and newly produced effects. This process of interaction is formalised using an R-residual interaction law between T and D.

B. C. d. S. Oliveira (Ed.): APLAS 2020, LNCS 12470, pp. 186–205, 2020.
https://doi.org/10.1007/978-3-030-64437-6_10

But there is more to say about running computations in interaction with environments from the "pragmatic" point of view: namely, how information is extracted from completed runs and how environments are prepared for runs.

We use algebras to interpret outcomes of completed runs, i.e., residual computations. Under favourable circumstances, algebras $\xi : RX \to X$ for a set of values X can be used to extract a single value from a computation over values; in this situation one often talks of them as *handlers* [16]. More generally, algebras can be used to observe or test computations, in terms of a set Z of observables (generalized truth values), as done, e.g., by Hasuo [5]. Algebras $\zeta : RZ \to Z$ lift value predicates $P : X \to Z$ to computation predicates $\zeta \circ RP : RX \to Z$.

Coalgebras $\chi : Y \to DY$, on the other hand, can be used to specify the environment that any initial state yields; an environment itself describing the response and state-change behaviour of a world a computation may be placed in. The carrier of such a coalgebra is the state set of the environment.

In this paper, we show that, given an algebra of R with carrier Z describing the handling of computations over observables, and a coalgebra of D with carrier Y describing the production of environments over states, we can turn an R-residual interaction law between T and D into a single algebra on T. This resulting algebra with carrier $Y \Rightarrow Z$ handles computations over state predicates, and can be used to describe, in one go, the behaviour of the whole system. This combination of tools gives rise to an isomorphism between the category of R-residual interaction laws between T and D and the category of *merge functors*, which merge coalgebras of D into algebras of R to form algebras of T.

We look at three running examples. One example is a computation which requests probabilistic weights from the environment for resolving nondeterministic choices. These weights are stored in the residual computation of weighted probabilistic choices. The second example incorporates an uncertain data reader, in which repeated state readings may be necessary before an effect is resolved, and each reading has an associated cost. We see this cost as an emergent effect resulting from the interaction, which we put in the residual computation. The last example combines probability with a comonadic model of global store.

There are many applications for this result on combining algebras, coalgebras and interaction laws. Firstly, it allows us to add program-environment interaction on top of pre-existing effect descriptions. For instance, if we have an algebra for an effect, e.g. a handler or some predicate lifting, it can be completed with an interaction law in order to add environments to the picture. This can, for instance, be seen in the last example, where we add global store to probability. Secondly, using algebras also enables us to describe more fully situations of program-environment interaction that cannot be analyzed in terms of an interaction law alone, as seen in the other two running examples. We thus obtain a flexible framework for describing, and potentially implementing, program-environment interactions.

The running examples are implemented using algebraic effects in the sense of Plotkin and Power [14,15] (see also Bauer [3]), which use effect operations from a signature that can be encoded by a functor F. To easily construct interaction

laws, we use *functor-comonad interaction laws*. Such a law between F and D can be extended to an interaction law between the free monad on F and comonad D via an isomorphism.

Two additional descriptions lie between interaction laws and merge functors. Combining interaction laws only with coalgebras yields *stateful runners* [7,20], and combining them only with algebras yields a novel concept of *continuation-based* runners, lifting an environment over value predicates to a single predicate on computations over values. Both types of runners can also be described as monad morphisms.

The next two sections give some preliminaries, where Sect. 2 focusses on formulating handler algebras and producer coalgebras, and Sect. 3 studies interaction laws. Section 4 introduces merge functors, and Sect. 5 establishes their isomorphism with interaction laws. Section 6 presents a way to formulate interaction laws locally on effect operations, and lastly, Sect. 7 talks about stateful and continuation-based runners.

We introduce the categorical concepts necessary for following the main story of the paper, but side remarks use more advanced category theory. Throughout the paper, we work with one Cartesian closed base category $\mathcal{C}$. The examples are all for **Set**. We write $Y \Rightarrow Z$ and occasionally Z^Y for exponents.

2 Effect Handling and Coeffect Production

2.1 Effect Handling

Effectful programs may produce multiple return values, no return values, or different return values in different situations, and they may communicate information to the environment. Following Moggi [10], the behaviours of such programs are abstracted into effectful computations. Effectful computations over a set of return values X are elements of the set TX for T some monad. A computation represents the behaviours of the program in terms of the requests it makes to the world (effects) it is run in and values it eventually can return depending on how the requests are responded.

Given a computation $t \in TX$ over the set of return values X, we may want to retrieve a single return value. To this end, we can use a monad algebra of T.

Definition 1. *An algebra $\xi : TX \to X$ of the underlying functor of a monad $T = (T, \eta^T, \mu^T)$ is said to be a* monad algebra *if it satisfies the following equations:*

$$\begin{array}{ccc} X & & \\ {\scriptstyle \eta^T_X}\downarrow & \searrow & \\ TX & \xrightarrow[\xi]{} & X \end{array} \qquad \begin{array}{ccc} TTX & \xrightarrow{T\xi} & TX \\ {\scriptstyle \mu^T_X}\downarrow & & \downarrow{\scriptstyle \xi} \\ TX & \xrightarrow[\xi]{} & X \end{array}$$

We denote the category of monad algebras of T by $\mathbf{Alg}(T)$. The Kleisli category $\mathbf{Kl}(T)$ is isomorphic to the full subcategory of $\mathbf{Alg}(T)$ given by monad algebras $\mu^T_X : T(TX) \to TX$ (the free algebras).

Besides directly extracting a return value from a computation, we can also use algebras to make observations about computations. Suppose that X is some set of values, Z is a set of observables, perhaps (generalized) truth values, and that RX for R some monad is the set of effectful computations over these values. Given a morphism $P : X \to Z$ that assigns observables to values (is a value predicate), an algebra $\zeta : RZ \to Z$ associates an observable to any given effectful computation via $\zeta \circ RP : RX \to Z$ (is a computation predicate).

Consider, for instance, the distributions monad $\mathcal{D}$. A randomized computation is represented by an element $t \in \mathcal{D}X$ for some value set X. Not every set X can be (meaningfully) endowed with an algebra structure $\xi : \mathcal{D}X \to X$: there may be no way of combining the many possible return values $x \in X$ appearing in a computation $t \in \mathcal{D}X$ into a single value.

A solution to this issue is to work with observations. We use a set of observables Z, which in this case we can take to be probabilities, the real number interval from 0 to 1 denoted by $[0, 1]$. Given some predicate on return values $P : X \to Z$ (assigning a probability to each $x \in X$), we can transform $t \in \mathcal{D}X$ into $(\mathcal{D}P)(t) \in \mathcal{D}[0, 1]$. We then use an algebra $\mathsf{E} : \mathcal{D}[0, 1] \to [0, 1]$ for calculating *expectations* to compute an observed probability for t.

We can take a more syntactical approach, describing effects using algebraic effect operations [15]. We consider a signature of effect operations Σ, each operation op having an arity given by an object of the category $ar(\mathsf{op}) \in \mathcal{C}$. This arity tells us how many possible continuations there are for a program when this effect is encountered. For instance, a binary choice operation would have arity 2, meaning there are two continuations.

Given such a signature Σ, we generate the free monad $T_\Sigma X := \mu V.X + \sum_{\mathsf{op} \in \Sigma}(ar(\mathsf{op}) \Rightarrow V)$. Given $f : ar(\mathsf{op}) \Rightarrow T_\Sigma X$, we write $\mathsf{op}(f) \in T_\Sigma X$ for the appropriate element in $T_\Sigma X$. If $ar(\mathsf{op}) = n = \{0, \ldots, n-1\}$ and $t_0, \ldots, t_{n-1} \in T_\Sigma X$, we may alternatively write $\mathsf{op}(t_0, \ldots, t_{n-1})$.

Example 1 (Probabilistic weights). Consider computations which can, for each $q \in [0, 1]$, have a q-weighted probabilistic binary choice or_q with $ar(\mathsf{or}_q) = \mathbf{2}$. Let $\Sigma := \{\mathsf{or}_q \mid q \in [0, 1]\}$, with computations over Z living in the free monad $RZ := T_\Sigma Z$ of binary leaf trees with nodes labelled with weights. As observables, we take expectations of truth $[0, 1]$, and we inductively define our algebra $\mathsf{Exp} : R[0, 1] \to [0, 1]$ by $\mathsf{Exp}(\mathsf{or}_q(x, y)) := (1 - q)\mathsf{Exp}(x) + q\mathsf{Exp}(y)$.

Example 2 (Cost of computations). We consider a simpler example where we associate a *cost* to certain computation steps. This can for instance represent time investment, or expenditure of some other resource like memory or energy. We consider a single tick operation $\Sigma := \{\mathsf{tick}\}$ with arity 1, and allow computation to continue forever: $RZ := \nu W.Z + W$. This monad is given by final coalgebras, not initial algebras, i.e., is not the free monad on the identity functor, but the free completely iterative monad in the sense of Aczel et al. [1], informally, the smallest monad that supports both the tick operation and (guarded) iteration. As an algebra, we take the cost tallying device $\mathsf{Tal} : R(\mathbb{N}_\infty) \to \mathbb{N}_\infty$ given by $\mathsf{Tal}(\mathsf{tick}(t)) = 1 + \mathsf{Tal}(t)$ for finite sequences of ticks and $\mathsf{Tal}(t) = \infty$ for t the infinite sequence of ticks.

2.2 Coeffect Production

On the other side of the story, we consider environments. They react to requests of the computation, but are otherwise passive. An environment reacts to requests by responding, and it also has a state that it changes as it responds. Such a process is called coeffectful. Notions of coeffect can be modelled with a comonad D. Environments over a state set Y are elements of DY; an environment is a description of the response and state-change behaviour of the world, including an initial state, which can be extracted using the counit ε_Y.

Most of the work on coeffects modelled by comonads has concentrated on scenarios where a notion of computation is coeffectful by primarily relying on coeffect *consumption* (coeffect *cooperations*); computations are modelled by co-Kleisli arrows, e.g., [12,22]. Typical examples of this are computations with causal stream functions (dataflow computation) and stencil computations (a.k.a. cellular automata) [4,21]. Here, in contrast, we are interested not in coeffectful computations, but in coeffectful environments. Central for us in this endeavour are coeffect *producers*, which are coalgebras of the comonad. They assign to every initial state drawn from a fixed state set an environment in a consistent way: given an environment assigned to some initial state by a coeffect producer, its continuation from any point must be obtainable by applying the coffect producer to the state reached by that point as the new initial state. See, e.g., [13,18,20] for examples.

Definition 2. *An coalgebra $\chi : Y \to DY$ of the underlying functor of a comonad $D = (D, \varepsilon, \delta)$ is called a* comonad coalgebra *if it satisfies the following equations:*

$$\begin{array}{ccc} Y & & \\ {\scriptstyle \varepsilon_Y}\uparrow & \searrow & \\ DY & \xleftarrow[\chi]{} & Y \end{array} \qquad \begin{array}{ccc} DDY & \xleftarrow{D\chi} & DY \\ {\scriptstyle \delta_Y}\uparrow & & \uparrow{\scriptstyle \chi} \\ DY & \xleftarrow[\chi]{} & Y \end{array}$$

Let $\mathbf{Coalg}(D)$ be the category of comonad coalgebras on D. The coKleisli category $\mathbf{CoKl}(D)$ is isomorphic to the full subcategory of $\mathbf{Coalg}(D)$ given by comonad coalgebras $\delta_Y : DY \to D(DY)$ (the cofree coalgebras).

To facilitate the description of examples, we can again take a more syntactical approach. Typically, a coeffect is described using coalgebraic cooperations. We consider a signature Π of such cooperations, each $\mathsf{cop} \in \Pi$ having an arity $ar(\mathsf{cop}) \in \mathcal{C}$. The arity gives the range of responses the environment can give.

The state of the environment contains for each cooperation a particular response and a new environment. Given a signature Π, we consider the cofree comonad $D_\Pi Y := \nu W.\, Y \times \Pi_{\mathsf{cop} \in \Pi}(ar(\mathsf{cop}) \times W)$. Supposing $e \in D_\Pi Y$ and $\mathsf{cop} \in \Pi$, we have that $\mathsf{cop}(e) \in ar(\mathsf{cop}) \times D_\Pi Y$ is a pair (c, e') consisting of a piece of data c provided by the environment and the continuation of the environment e'.

Example 3 (Stream of data). Given some object of data C, we consider an environment which can supply a *stream* of datapoints from C. We take one cooperation $\Pi := \{\mathsf{give}\}$ of arity C, and consider the comonad $DY := D_\Pi Y \cong$

$\nu W.\, Y \times (C \times W) \cong (Y \times C)^{\mathbb{N}}$ of streams over $Y \times C$. As environment we take $E := C^{\mathbb{N}} \cong D1$. The *producer* is the cofree coalgebra $\delta_1 : E \to DE$, which sends a stream σ to $\delta_1(\sigma)$ where $\mathsf{give}(\delta_1(\sigma)) := (\sigma(0), \delta_1(\lambda n.\, \sigma(n+1)))$.

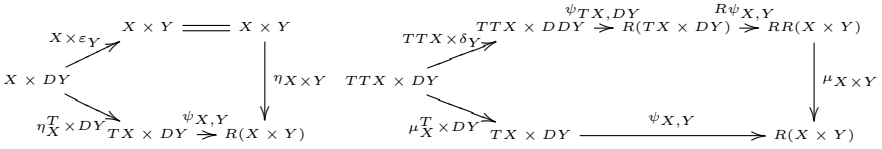
Example 4 (Global store). We consider a set of data C, and an environment which has one datapoint from C stored in its memory. We have $\Pi := \{\mathsf{give}\} \cup \{\mathsf{change}_c \mid c \in C\}$, where $ar(\mathsf{give}) := C$ and $ar(\mathsf{change}_c) := 1$. We can take $DY := D_\Pi Y \cong \nu W.Y \times (C \times W) \times (1 \times W)^C$ to be the cofree comonad, which allows for giving and receiving data C. A producer for a *global store* environment generates from a global state the appropriate environment which acts in the following way: (1) when data is provided on request, the internal state does not change, and (2) when data is received, the environment changes its internal state accordingly. We formulate the producer $\mathsf{GS} : Y \to DY$ with $Y := C$ where $\mathsf{give}(\mathsf{GS}(c)) := (c, \mathsf{GS}(c))$ and $\mathsf{change}_d(\mathsf{GS}(c)) := (*, \mathsf{GS}(d))$. A smaller comonad defined by $D'Y := C \times (C \Rightarrow Y)$ allows only producers that obey the coequations of global store; these amount to arrays, a.k.a. lenses. The set $D'Y$ consists of the elements of DY satisfying the coequations; as seen in the literature [13,18].

3 Interaction Laws

We now formulate how computations can interact with environments, with coeffects reacting to effects. Supposing we have the effects of interest described by some monad T, and the coeffects by a comonad D, an *interaction law* between T and D tells us how coeffects can be used to resolve effects.

In general, not all effects of a computation may be resolved by the environment it is run against. Moreover, the interaction between effects and coeffects may produce new effects. We therefore use another monad $R = (R, \eta, \mu)$ for residual effects. We study R-residual interaction laws of T and D by Katsumata et al. [7]. They are an elaboration of ideas and abstraction of concepts by Plotkin and Power [13] and Møgelberg and Staton [9].

Definition 3. *An* R-residual interaction law *of* $T = (T, \eta^T, \mu^T)$ *and* $D = (D, \varepsilon, \delta)$ *is given by a natural transformation typed* $\psi^{(1)}_{X,Y} : TX \times DY \to R(X \times Y)$ *satisfying the (co)unit and (co)multiplication agreement equations*



By the Yoneda lemma, a natural transformation $\psi^{(1)}$ above can be alternatively given as a natural transformation typed [1]

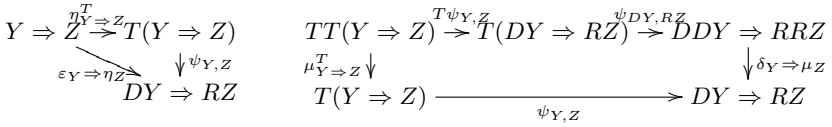
$$\psi^{(0)}_{X,Y,Z} : \mathcal{C}(X \times Y, Z) \to \mathcal{C}(TX \times DY, RZ)$$

[1] Note that this is not the same in general as to have a natural transformation typed $X \times Y \Rightarrow Z \to TX \times DY \Rightarrow RZ$.

and therefore by Currying and symmetry also by natural transformations

$$\psi^{(2)}_{X,Z} : D(X \Rightarrow Z) \to TX \Rightarrow RZ, \qquad \psi^{(3)}_{Y,Z} : T(Y \Rightarrow Z) \to DY \Rightarrow RZ.$$

In this paper, we use all these formats, especially the 3rd in the next few sections. Translating the equations of interaction laws into the 3rd format, we get:



We will write $\mathbf{MCIL}_R(T, D)$ for the set of R-residual interaction laws between T and D (ignoring the exact format chosen).

The intuition for interaction laws is as follows. In the 0th format, X is the set of return values of computations, Y is the state set of environments and Z is the set of observables (or truth values). An interaction law ψ says that, as soon as we know how to observe a value-state pair, a computation over values and an environment can be combined to yield a computation over observables. It must be natural in X, Y, Z to reflect that interactions only pass values, states and observables around, but do not inspect them. In the 1st format, the observables are $X \times Y$, i.e., value-state pairs are directly observable. In the 2nd format, the states of environments are $X \Rightarrow Z$, i.e., value predicates. In the 3rd format, the values that given computations return are $Y \Rightarrow Z$, i.e., state predicates.

Later in the paper, we will use algebras and coalgebras to explain what interaction laws do in different terms.

Example 5 (Probabilistic weight requester). In this example, computations may have to make certain binary choices. They are represented by binary trees $TX := T_{\Sigma'}X$ where Σ' has one operation or of arity $\mathbf{2}$. To make a nondeterministic choice, a computation requests a probabilistic weight from its environment. This environment is given by a stream DY of such weights as in Example 3, using as data object $C := [0, 1]$. Having received a weight for each choice, it generates a tree of probabilistic choices: $RZ := T_\Sigma Z$ as in Example 1. This is done using an *interaction law* $\psi_{Y,Z} : T(Y \Rightarrow Z) \to DY \Rightarrow RZ$ where:

– $\psi_{Y,Z}(\mathsf{or}(a, b))(e) := \mathsf{or}_q(\psi_{Y,Z}(a)(e'), \psi_{Y,Z}(b)(e'))$, if $(q, e') = \mathsf{give}(e)$.

Example 6 (Uncertain stream reader). We use DY as in Example 3, and consider a computation which can request datapoints (elements of a set C) from its environment. Programs use one effect operation $\Sigma' := \{\mathsf{get}\}$ with $ar(\mathsf{get}) := C$ and $TX := T_{\Sigma'}X$. Upon a request, the environment will keep giving datapoints until it gives the same datapoint twice in a row. We associate to each give a cost, which we store with a tick in the residual computation in $RZ := \nu W.\, Z + W$, as given in Example 2. We describe this multistep protocol with the interaction law $\psi_{Y,Z}$ as follows: given $e \in DY$, with $(c_0, e_0) := \mathsf{give}(e)$, and $(c_1, e_1) := \mathsf{give}(e_0)$:

$$– \ \psi_{Y,Z}(\mathsf{get}(f))(e) := \begin{cases} \mathsf{tick}(\mathsf{tick}(\psi_{Y,Z}(f(c_0))(e_1))) & \text{if } c_0 = c_1, \\ \mathsf{tick}(\psi_{Y,Z}(\mathsf{get}(f))(e_0)) & \text{if } c_0 \neq c_1\ . \end{cases}$$

Example 7 (Combining global store with probability). Lastly, we consider a more traditional example, where some but not all effects are resolved, and no new effects are generated. Take Σ and Π from Examples 1 and 4 respectively, and let $\Sigma' := \Sigma + \{\mathsf{lookup}, \mathsf{update}_c \mid c \in C\}$ where $ar(\mathsf{lookup}) := C$ and $ar(\mathsf{update}_c) := 1$. We take computations which can request and update a global store, and make probabilistic choices, denoted by $TX := T_{\Sigma'}X$. As comonad we use the environment $DY := D_\Pi Y$, and as residual monad $RZ := R_\Sigma Z$ the weighted choice trees. The interaction law $\psi_{Y,Z}$ resolves only the global store requests. For $e \in DY$, let $(c, e') := \mathsf{give}(e)$, and for each $d \in C$ let $(*, e_d) := \mathsf{change}_d(e)$, then

- $\psi_{Y,Z}(\mathsf{lookup}(f))(e) := \psi_{Y,Z}(f(c))(e')$,
- $\psi_{Y,Z}(\mathsf{update}_d(t))(e) := \psi_{Y,Z}(t)(e_d)$,
- $\psi_{Y,Z}(\mathsf{or}_q(a,b))(e) := \mathsf{or}_q(\psi_{Y,Z}(a)(e), \psi_{Y,Z}(b)(e))$.

In Sect. 6, we discuss a method for showing that the above constructions satisfy the unit and multiplication equations for interaction laws.

Katsumata et al. [7] proved that R-residual interaction laws of T, D are in a bijection with monad morphisms from T to the monad $D \multimap R$ where $D \multimap -$ is the right adjoint of $- \star D$ and $\star$ is the Day convolution. This monad is explicitly given by $(D \multimap R)X = \int_Y DY \Rightarrow R(X \times Y) \cong \int_{Y,Z} \mathcal{C}(X \times Y, Z) \pitchfork (DY \Rightarrow RZ) \cong \int_Z D(X \Rightarrow Z) \Rightarrow RZ$ (with $\int$ with subscript for ends and $\pitchfork$ for powers).

A *morphism* between two interaction laws (T, D, R, ψ) and (T', D', R', ψ') is given by (co)monad morphisms $t : T \to T'$, $d : D' \to D$ and $r : R \to R'$ satisfying the left equation below for the 1st format and the right equation for the 3rd format:

$$\begin{array}{ccc} TX \times DY & \xrightarrow{\psi_{X,Y}} & R(X \times Y) \\ {\scriptstyle TX \times d_Y}\uparrow & & \downarrow{\scriptstyle r_{X\times Y}} \\ TX \times D'Y & & \\ {\scriptstyle t_X \times D'Y}\downarrow & & \\ T'X \times D'Y & \xrightarrow{\psi'_{X,Y}} & R'(X \times Y) \end{array} \qquad \begin{array}{ccc} T(Y \Rightarrow Z) & \xrightarrow{\psi_{X,Y}} & DY \Rightarrow RZ \\ {\scriptstyle t_{Y\Rightarrow Z}}\downarrow & & \downarrow{\scriptstyle d_Y \Rightarrow r_Z} \\ T'(Y \Rightarrow Z) & \xrightarrow{\psi'_{X,Y}} & D'Y \Rightarrow R'Z \end{array}$$

(Note the direction of d.) Interaction laws form a category. The bijection with monad morphisms extends to an isomorphism of categories, see [7].

4 Merge Functors

We are interested in how interaction laws can be combined with algebras for handling residual effects and coalgebras for producing coeffects. In general, we get what we call a merge functor. This merges a coalgebra into an algebra creating a new algebra.

Definition 4. *A* merge functor *for T, D, R is given by a functor $M : (\mathbf{Coalg}(D))^{\mathrm{op}} \times \mathbf{Alg}(R) \to \mathbf{Alg}(T)$ which is carrier-exponentiating:*

$$\begin{array}{ccc} (\mathbf{Coalg}(D))^{\mathrm{op}} \times \mathbf{Alg}(R) & \xrightarrow{M} & \mathbf{Alg}(T) \\ {\scriptstyle U^{\mathrm{op}} \times U}\downarrow & & \downarrow{\scriptstyle U} \\ \mathcal{C}^{\mathrm{op}} \times \mathcal{C} & \xrightarrow{\Rightarrow} & \mathcal{C} \end{array}$$

U are the relevant forgetful functors, which are the left (resp. right) adjoints of the co-Eilenberg-Moore (resp. Eilenberg-Moore) adjunctions of D (resp. R, T).

Note in particular the three conditions which need to hold for M to be a merge functor:

- Every functor algebra in the image of M needs to be a monad algebra;
- M needs to be functorial in its comonad coalgebra and monad algebra arguments, sending coalgebra and algebra morphisms to an algebra morphism;
- On the level of carriers, M needs to be the exponentiation function.

Here is a variation of merge functors. A *Kleisli merge functor* for T, D, R is a functor $N : (\mathbf{CoKl}(D))^{\mathrm{op}} \times \mathbf{Kl}(R) \to \mathbf{Alg}(T)$ which is carrier-exponentiating in the sense that

$$\begin{array}{ccc} (\mathbf{CoKl}(D))^{\mathrm{op}} \times \mathbf{Kl}(R) & \xrightarrow{N} & \mathbf{Alg}(T) \\ \downarrow{\scriptstyle F^{\mathrm{op}} \times F} & & \downarrow{\scriptstyle U} \\ \mathcal{C}^{\mathrm{op}} \times \mathcal{C} & \xrightarrow{\Rightarrow} & \mathcal{C} \end{array}$$

where $F : \mathbf{CoKl}(D) \to \mathcal{C}$ is the left adjoint of the coKleisli adjunction of D and $F : \mathbf{Kl}(R) \to \mathcal{C}$ is the right adjoint of the Kleisli adjunction of R.

Here and in the rest of this paper, when we refer to $\mathbf{CoKl}(D)$, we mean the full subcategory of $\mathbf{Coalg}(D)$ given by the cofree coalgebras, which is isomorphic, and similarly for $\mathbf{Kl}(R)$ and the full subcategory of $\mathbf{Alg}(R)$ given by the free algebras. Under this view, the two functors F are still forgetful functors.[2]

Proposition 1. *Any Kleisli merge functor has a unique extension to merge functors. This gives us a bijection between the sets of merge functors and Kleisli merge functors for T, D, R:*

$$\frac{(\mathbf{CoKl}(D))^{\mathrm{op}} \times \mathbf{Kl}(R) \to_{\mathrm{ce.}} \mathbf{Alg}(T)}{(\mathbf{Coalg}(D))^{\mathrm{op}} \times \mathbf{Alg}(R) \to_{\mathrm{ce.}} \mathbf{Alg}(T)}$$

(where 'ce.' stands for carrier-exponentiating).

To see why this is, let us observe the following. Suppose we have a merge functor $M : \mathbf{Coalg}(D)^{\mathrm{op}} \times \mathbf{Alg}(R) \to \mathbf{Alg}(T)$. For any comonad coalgebra $\chi : Y \to DY$ and any monad algebra $\zeta : RZ \to Z$, by functoriality of M and the counit and unit equations of χ and ζ, we have

$$\begin{array}{ccccc} T(Y \Rightarrow Z) & \xrightarrow{T(\varepsilon_Y \Rightarrow \eta_Z)} & T(DY \Rightarrow RZ) & \xrightarrow{M(\delta_Y, \mu_Z)} & DY \Rightarrow RZ \\ & \searrow & \downarrow{\scriptstyle T(\chi \Rightarrow \zeta)} & & \downarrow{\scriptstyle \chi \Rightarrow \zeta} \\ & & T(Y \Rightarrow Z) & \xrightarrow{M(\chi, \zeta)} & Y \Rightarrow Z \end{array}$$

This uses functoriality of M on the facts that χ is a coalgebra morphism from χ to δ_Y and that ζ is an algebra morphism from μ_Z to ζ, which are consequences of the comultiplication and multiplication equations of χ and ζ.

[2] Namely, they send δ_Y^D and μ_Z^R to DY and RZ respectively.

So any merge functor is determined by its Kleisli merge sub-functor. We therefore have but one candidate for extending a given Kleisli merge functor $N : (\mathbf{CoKl}(D))^{\mathrm{op}} \times \mathbf{Kl}(R) \to \mathbf{Alg}(T)$ into a merge functor $\widehat{N}$, which is: $\widehat{N}(\chi, \zeta) = (\chi \Rightarrow \zeta) \circ N(\delta_Y, \mu_Z) \circ T(\varepsilon_Y \Rightarrow \eta_Z)$. It is easy to show that $\widehat{N}$ is functorial and that the functor algebras it delivers are monad algebras.

5 The Interaction Law, Merge Functor Isomorphism

Given an interaction law ψ, we define a Kleisli merge functor M_ψ as follows. For a cofree coalgebra $\delta_Y : DY \to DDY$ and a free algebra $\mu_Z : RRZ \to RZ$, we construct an algebra

$$M_\psi(\delta_Y, \mu_Z) := T(DY \Rightarrow RZ) \xrightarrow{\psi_{DY,RZ}} DDY \Rightarrow RRZ \xrightarrow{\delta_Y \Rightarrow \mu_Z} DY \Rightarrow RZ.$$

M_ψ is easily seen to be functorial and delivering monad algebras.

The construction $M_{(-)}$ gives rise to the following coincidence.

Proposition 2. *There is a bijection between R-residual interaction laws of T and D, and Kleisli merge functors for T, D, R:*

$$\frac{\mathbf{MCIL}_R(T, D)}{(\mathbf{CoKl}(D))^{\mathrm{op}} \times \mathbf{Kl}(R) \to_{\mathrm{ce.}} \mathbf{Alg}(T)}$$

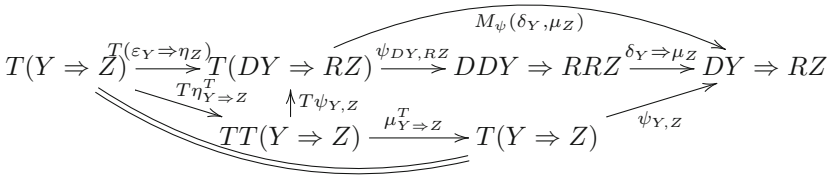
We need to show that the construction $M_{(-)}$ gives a bijection. We do this by explicitly defining the inverse. Given a Kleisli merge functor $M : (\mathbf{CoKl}(D))^{\mathrm{op}} \times \mathbf{Kl}(R) \to \mathbf{Alg}(T)$, we construct a natural transformation

$$\psi^M_{Y,Z} := T(Y \Rightarrow Z) \xrightarrow{T(\varepsilon_Y \Rightarrow \eta_Z)} T(DY \Rightarrow RZ) \xrightarrow{M(\delta_Y, \mu_Z)} DY \Rightarrow RZ$$

It is easy to verify that ψ^M fulfills the conditions of an interaction law.

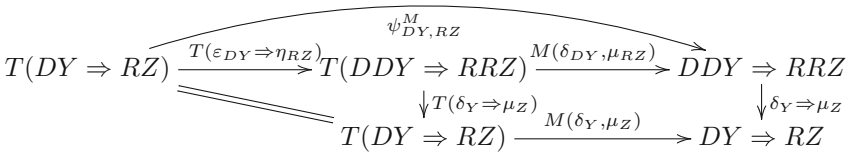
Lemma 1. *The construction $\psi^{(-)}$ is an inverse to the construction $M_{(-)}$.*

Proof. We show that $\psi^{M_\psi} = \psi$.



The diagram commutes by the definition of M_ψ, the (co)unit and (co)multiplication equations of ψ, and right unitality of T. As the path at the top is the constructed interaction law ψ^{M_ψ}, and that at the bottom is the given interaction law ψ, the two coincide.

We show that $M^{\psi_M} = M$:



The diagram commutes by the definition of ψ^M, left unitality of D and R and functoriality of M applied to the facts that δ_Y and μ_Z are (co)algebra morphisms. Following the top path, we get the constructed merge function M_{ψ^M}, whereas following the bottom path yields the given merge function M. We conclude that the two coincide. □

This finishes the proof of Proposition 2. Combining this with Proposition 1, we have proved the following.

Corollary 1. *There is a bijection between R-residual interaction laws of T, D and merge functors for T, D, R.*

Explicitly, the bijection of Corollary 1 sends an interaction law ψ to $M_\psi : (\mathbf{Coalg}(D))^{\mathrm{op}} \times \mathbf{Alg}(R) \to \mathbf{Alg}(T)$ which does:

$$M_\psi(\chi : Y \to DY, \zeta : RZ \to Z) := \; T(Y \Rightarrow Z) \xrightarrow{\psi_{Y,Z}} DY \Rightarrow RZ \xrightarrow{\chi \Rightarrow \zeta} Y \Rightarrow Z \quad .$$

Example 8 (Probabilistic weight requester). We use the interaction law ψ from Example 5 to merge the coalgebra δ_1 from Example 3 into the algebra Exp from Example 1. We get $M_\psi(\delta_1, \mathsf{Exp}) : T(E \Rightarrow [0,1]) \to E \Rightarrow [0,1]$, with as carrier set the $[0,1]$-valued predicates on streams $E = [0,1]^{\mathbb{N}}$. Suppose we have some predicate $P : X \to E \Rightarrow [0,1]$ giving some expectation to each return value and final state. Then, given some computation $t \in TX$, we can find the weakest precondition $M_\psi(\delta_1, \mathsf{Exp})(TP(t)) \in E \Rightarrow [0,1]$, which gives, for each initial state e of the environment, the expectation of the computation, determined by the postcondition P on the return value and the final states yielded.

Example 9 (Uncertain stream reader). We use the interaction law ψ from Example 6 to merge the coalgebra δ_1 from Example 3 into the algebra Tal from Example 2. We get $M_\psi(\delta_1, \mathsf{Tal}) : T(E \Rightarrow \mathbb{N}_\infty) \to E \Rightarrow \mathbb{N}_\infty$, with as carrier set $\mathbb{N}_\infty$-valued predicates on streams $E = C^{\mathbb{N}}$. This merged algebra computes, for each initial state of the environment, how many responses the environment gives during the interaction, and adds it to the perceived value of the final state. Streams which behave unreliably will naturally give more datapoints, as they create more uncertainty. For instance, the stream $10000\ldots$ will make a program return the same values as the stream $0000\ldots$, but it may invoke more ticks (one more tick) than the latter. So, for each $t \in T(E \Rightarrow \mathbb{N}_\infty)$, $M_\psi(\delta_1, \mathsf{Tal})(t)(10000\ldots) \geq M_\psi(\delta_1, \mathsf{Tal})(t)(0000\ldots)$.

Example 10 (Combining global store with probability). Lastly, we use the interaction law ψ from Example 7 to merge GS from Example 4 into Exp from Example 1. We retrieve the algebra $M_\psi(\mathsf{GS}, \mathsf{Exp}) : T(C \Rightarrow [0,1]) \to C \Rightarrow [0,1]$ from previous work [23], whose carrier set is given by $[0,1]$-valued store predicates.

We have not yet specified what a morphism between merge functors is. We define them so that they will coincide with morphisms between interaction laws. We say that a morphism between two merge functors (T, D, R, M) and (T', D', R', M') is a triple of (co)monad morphisms $t : T \to T'$, $d : D' \to D$ and $r : R \to R'$ such that, for any comonad coalgebra $\chi : Y \to D'Y$ and any monad algebra $\zeta : R'Z \to Z$, $M'(\chi, \zeta) \circ t_{Y \Rightarrow Z} = M(d_Y \circ \chi, \zeta \circ r_Z)$.

Proposition 3. *A triple (t, d, r) of (co)monad morphisms forms a morphism between interaction laws ψ and ψ' if and only if it is a morphism between merge functors M_ψ and $M_{\psi'}$.*

Corollary 2. *The category of residual interaction laws is isomorphic to the category of merge functors, thus preserving the underlying (co)monads.*

Since the isomorphism preserves the underlying (co)monads of its objects, we can also fix some of T, D, and R, and the isomorphism still holds. For instance, we get an isomorphism between R-residual interaction laws for some fixed R, and the category of merge functors for the same fixed R, with D and T varying.

6 Interaction Laws for Free Monads

One thing we have not yet done is show that the interaction laws of the examples satisfy the unit and multiplication equations. This is often tedious to do in practice. In this section, we discuss a recipe for generating interaction laws when T is a free monad. This recipe is exhaustive and without redundancy: it generates all interaction laws exactly once. We start with a general Cartesian closed category $\mathcal{C}$ first, and do some further simplifications for **Set** later on.

Given a functor F, the underlying functor T of the free monad on F is given by initial algebra carriers: $TX := \mu V.X + FV$. The structure maps $X + FTX \to TX$ split into $\eta^T_X : X \to TX$ and $\sigma_X : FTX \to TX$. The unit is η^T_X and the multiplication μ^T_X is the unique solution to the initial algebra diagram:

$$\begin{array}{ccccc} TX & \xrightarrow{\eta^T_{TX}} & TTX & \xleftarrow{\sigma_{TX}} & FTTX \\ & \searrow & \downarrow{\scriptstyle \mu^T_X} & & \downarrow{\scriptstyle F\mu^T_X} \\ & & TX & \xleftarrow{\sigma_X} & FTX \end{array}$$

Now R-residual interaction laws between D and T can be defined in "small steps", in terms of F, giving rise to the following result.

Proposition 4. *If T is the free monad on F, then there is a bijection between R-residual interaction laws of T, D and natural transformations typed $\phi_{Y,Z} : F(Y \Rightarrow Z) \to DY \Rightarrow RZ$ (subject to no equations!).*

The natural transformation $\phi_{Y,Z}$ given above can also be seen as a *functor-comonad interaction law*; an intermediate between functor-functor and monad-comonad interaction laws of Katsumata et al. [7].

Given a functor-comonad interaction law $\phi_{Y,Z}$, we construct a natural transformation $\psi_{Y,Z}$ as the unique solution of the following initial algebra diagram:

$$\begin{array}{ccccc} Y \Rightarrow Z & \xrightarrow{\eta^T_{Y \Rightarrow Z}} & T(Y \Rightarrow Z) & \xleftarrow{\quad\sigma_{Y \Rightarrow Z}\quad} & FT(Y \Rightarrow Z) \\ & \searrow^{\varepsilon_Y \Rightarrow \eta_Z} & \downarrow{\scriptstyle \psi_{Y,Z}} & & \downarrow{\scriptstyle F\psi_{Y,Z}} \\ & & DY \Rightarrow RZ & \xleftarrow{\delta_Y \Rightarrow \mu_Z} DDY \Rightarrow RRZ & \xleftarrow{\phi_{DY,RZ}} F(DY \Rightarrow RZ) \end{array}$$

The natural transformation ψ satisfies the equations of a monad-comonad interaction law. In the reverse direction, we extract from a given monad-comonad interaction law ψ a natural transformation ϕ as follows:

$$\phi_{Y,Z} := \; F(Y \Rightarrow Z) \xrightarrow{F\eta^T_{Y \Rightarrow Z}} FT(Y \Rightarrow Z) \xrightarrow{\sigma_{Y \Rightarrow Z}} T(Y \Rightarrow Z) \xrightarrow{\psi_{Y,Z}} DY \Rightarrow RZ$$

Combining the proposition with Corollary 1, we get a corollary exploiting that the category $\mathbf{Alg}(T)$ is isomorphic to $\mathbf{alg}(F)$.

Corollary 3. *There is a bijection between R-residual functor-comonad interaction laws of F and D and functors $(\mathbf{Coalg}(D))^{\mathrm{op}} \times \mathbf{Alg}(R) \to \mathbf{alg}(F)$ that are exponentiation on the level of carriers.*

In the particular case of our examples, where $FX := \sum_{\mathsf{op} \in \Sigma}(ar(\mathsf{op}) \Rightarrow X)$, the functor-comonnad interaction law ϕ required for specifying the monad-comonad interaction law ψ decomposes, for each effect operation $\mathsf{op} \in \Sigma$, into a transformation $ar(\mathsf{op}) \Rightarrow (Y \Rightarrow Z) \to DY \Rightarrow RZ$ natural in Y and Z, an *operation-wise interaction law*. If R is strong and these natural transformations are strong in Z, which holds for our examples since they are in the category of sets, these natural transformations amount to transformations $\phi^{\mathsf{op}}_Y : DY \to R(ar(\mathsf{op}) \times Y)$ natural in Y for each operation op.

The transformation ϕ^{op}_Y specifies what happens when the operation op is encountered in the evaluation of some program. It tells us, given an environment, which effects are encountered, which continuation is chosen for the program, and what the new state is. The resulting interaction law ψ induced by our ϕ's given Proposition 4 will satisfy the following equation:

$$\psi_{Y,Z}(\mathsf{op}(t_1, \ldots, t_n))(e) \quad = \quad \mu_Z(R(\lambda(i, e').\psi_{Y,Z}(t_i)(e'))\ (\phi^{\mathsf{op}}_{DY}(\delta_Y(e)))) \qquad (1)$$

We show that the interaction laws for the examples satisfy the desired equations. This is done by specifying the natural transformations in such a way that the induced Eq. 1 coincides with the specification required in the examples.

For Example 5, where we have one operation or of arity $2 = \{0, 1\}$, we define: $\phi^{\mathsf{or}}_Y(e) := \mathsf{or}_q(\eta_Z(0, y), \eta_Z(1, y))$, where $(q, e') := \mathsf{give}(e)$ and $y := \varepsilon_Y(e')$. The transformation ϕ^{or}, which is obviously natural, tells us to allocate a probability of q to continue with 0, and $1 - q$ probability to continue with 1, and to finish in both cases in the next state, which is y.

The local transformation for Example 6 is slightly more involved. Here we have an effect operation get of arity C. The transformation $\phi_Y^{\mathsf{get}} : DY \to R(C \times Y)$ keeps applying give until the same data point comes out twice in a row (which may never happen) and returns the corresponding number of ticks and that data point and the final state (or an infinite sequence of ticks).

It is possible to design methods for defining interaction laws, using the above ideas. We would specify how to naturally generate three things from the environment: (1) which residual effects we get, (2) what piece of data is communicated to the program (the continuation), and (3) what is the state afterwards.

7 Runners

We have seen how an interaction law can be combined with a coalgebra of D and an algebra of R to yield an algebra of T. There are also intermediate constructions, combining the interaction law with only a coalgebra or an algebra. Depending on the choice, we get different results. Since they amount to monad morphisms from T to other monads, it is justified to call them runners.

7.1 Stateful Runners

Combining interaction laws with just coalgebras (rather than coalgebras and algebras) yields stateful runners in the sense of Uustalu [20].

An R-residual *stateful runner* of T for an object $Y \in \mathcal{C}$ is a natural transformation typed $\theta_X : TX \times Y \to R(X \times Y)$ subject to appropriate equations. With the appropriate concept of map, stateful runners make a category $\mathbf{Run}_R(T)$.

The following result gives alternative characterizations of stateful runners.

Proposition 5. *For any object $Y \in \mathcal{C}$, the following sets are in bijection:*

1. *R-residual stateful runners of T with carrier Y,*
2. *monad morphisms from T to St_Y^R, the R-transformed state monad for state set Y, defined by $\mathsf{St}_Y^R X := Y \Rightarrow R(X \times Y)$,*
3. *functors $\Theta : \mathbf{Alg}(R) \to \mathbf{Alg}(T)$ such that*

$$\begin{array}{ccc} \mathbf{Alg}(R) & \xrightarrow{\Theta} & \mathbf{Alg}(T) \\ {\scriptstyle U}\downarrow & & \downarrow{\scriptstyle U} \\ \mathcal{C} & \xrightarrow{Y \Rightarrow -} & \mathcal{C} \end{array}$$

The bijection between the first two items was pointed out in previous work [7,20]. These bijections extend to isomorphisms of the relevant total categories such as $\mathbf{Run}_R(T)$.

From the bijection between the 1st and 3rd item, by Corollary 1, we can conclude that interaction laws are in bijection with *D-coalgebraic specifications* of runners, which we define to be carrier-preserving functors $\Psi : \mathbf{Coalg}(D) \to \mathbf{Run}_R(T)$. Katsumata et al. [7] proved this bijection directly, rather than from Corollary 1 and Proposition 5, circumventing functors $\mathbf{Alg}(R) \to \mathbf{Alg}(T)$.

Explicitly, given an interaction law ψ (in the 1st format), the runner spec Ψ for comonad coalgebras $\chi : Y \to DY$ is given by

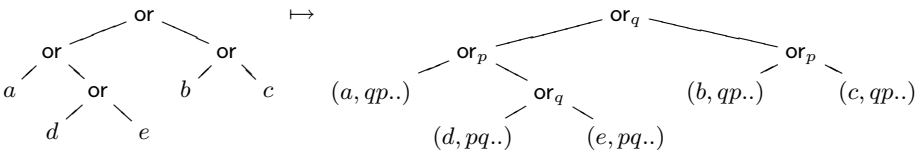
$$(\Psi\,\chi)_X := \; TX \times Y \xrightarrow{TX \times \chi} TX \times DY \xrightarrow{\psi_{X,Y}} R(X \times Y)\;.$$

Given a runner spec Ψ, the interaction law is defined by

$$\psi_{X,Y} := \; TX \times DY \xrightarrow{(\Psi\,\delta_Y)_X} R(X \times DY) \xrightarrow{R(X \times \varepsilon_Y)} R(X \times Y)\;.$$

Ahman and Bauer [2] defined runners of T as coalgebras of a specific comonad, namely the Sweedler dual of T with respect to R, studied in detail by Katsumata et al. [7]. That comonad is the greatest comonad that T interacts with R-residually. For that comonad, one has $\mathbf{Coalg}(D) \cong \mathbf{Run}_R(T)$, justifying this alternative definition. Runners of T for $R := \mathsf{Id}$ have also been called *coalgebras* of the monad T (notice: coalgebras, not algebras) [17].

Example 11 (Probabilistic weight requester). We look at Example 8 under the lens of stateful runners. Let Ψ be the coalgebraic specification of runners associated to ψ, and consider the runner $\Psi(\delta_1) : TX \times E \to R(X \times E)$. Given some computation $t \in TX$ and some state of the environment given by a stream of data $\sigma \in C^{\mathbb{N}}$, the runner produces some weighted choice tree $\Psi(\delta_1)(t, \sigma) \in T_\Sigma(X \times E)$. In this example, σ is used to label all the nodes of the initial tree t with the values of the stream σ. If the node has height n in the tree, it will be given label $\sigma(n)$. Each leaf of t will be joined with the remainder of the σ leftover after labelling. For instance, $\Psi(\delta_1)$ given stream $qpqpqpqp\ldots$ will send the following tree of TX to the given tree in $R(X \times E)$:



7.2 Continuation-Based Runners

If we combine interaction laws with algebras only, we get a novel concept of continuation-based runners.

We define a D-fuelled *continuation-based runner* of T for an object $Z \in \mathcal{C}$ to be a natural transformation typed $\theta_X : D(X \Rightarrow Z) \to TX \Rightarrow Z$ satisfying

$$\begin{array}{ccc} D(X \Rightarrow Z) & \xrightarrow{\theta_X} & TX \Rightarrow Z \\ & \searrow^{\varepsilon_{X \Rightarrow Z}} & \downarrow{\scriptstyle \eta^T_X \Rightarrow Z} \\ & & X \Rightarrow Z \end{array} \qquad \begin{array}{ccc} D(X \Rightarrow Z) & \xrightarrow{\quad\theta_X\quad} & TX \Rightarrow Z \\ {\scriptstyle \delta_{X \Rightarrow Z}}\downarrow & & \downarrow{\scriptstyle \mu^T_X \Rightarrow Z} \\ DD(X \Rightarrow Z) & \xrightarrow{D\theta_X} D(TX \Rightarrow Z) & \xrightarrow{\theta_{TX}} TTX \Rightarrow Z \end{array}$$

With the appropriate concept of map, D-fuelled continuation-based runners form a category $\mathbf{CRun}_D(T)$.

We make the following observation that also extends to isomorphisms of categories.

Proposition 6. *For any object $Z \in \mathcal{C}$, the following sets are in bijection:*

1. *D-fuelled continuation-based runners of T with carrier Z,*
2. *monad morphisms from T to Cnt_Z^D, the D-transformed continuation monad for answer set Z, defined by $\mathsf{Cnt}_Z^D X := D(X \Rightarrow Z) \Rightarrow Z$,*
3. *functors $\Theta : (\mathbf{Coalg}(D))^{\mathrm{op}} \to \mathbf{Alg}(T)$ such that*

$$\begin{array}{ccc} (\mathbf{Coalg}(D))^{\mathrm{op}} & \xrightarrow{\Theta} & \mathbf{Alg}(T) \\ {\scriptstyle U^{\mathrm{op}}}\downarrow & & \downarrow{\scriptstyle U} \\ \mathcal{C}^{\mathrm{op}} & \xrightarrow{-\Rightarrow Z} & \mathcal{C} \end{array}$$

It follows from Corollary 1 that R-residual T, D-interaction laws are in a bijection with *R-algebraic specifications* of D-fuelled continuation-based T-runners, by which we mean carrier-preserving functors $\Psi : \mathbf{Alg}(R) \to \mathbf{CRun}_D(T)$.

Explicitly, given an interaction law ψ in the 2nd format, the corresponding runner spec Ψ is defined by

$$(\Psi\,\zeta)_X := \; D(X \Rightarrow Z) \xrightarrow{\psi_{X,Z}} TX \Rightarrow RZ \xrightarrow{TX \Rightarrow \zeta} TX \Rightarrow Z$$

for monad algebras $\zeta : RZ \to Z$. In the reverse direction, given a runner spec Ψ, the interaction law ψ is

$$\psi_{X,Z} := \; D(X \Rightarrow Z) \xrightarrow{D(X \Rightarrow \eta_Z)} D(X \Rightarrow RZ) \xrightarrow{(\Psi\,\mu_Z)_X} TX \Rightarrow RZ \; .$$

Continuation-based runners can be understood as a predicate-lifting device: they lift an environment that has as states Z-valued predicates on values X to a Z-valued predicate on computations TX.

Example 12 (Uncertain stream reader). We look at Example 9 under the lens of continuation-based runners. Let Ψ be the algebraic specification of runners associated to ψ, and consider the runner $\Psi(\mathsf{Tal}) : D(X \Rightarrow \mathbb{N}_\infty) \to TX \Rightarrow \mathbb{N}_\infty$. Take $P \in D(X \Rightarrow \mathbb{N}_\infty)$ to be some environment over value predicates as states, which can be expressed as an element of $((X \Rightarrow \mathbb{N}_\infty) \times C)^{\mathbb{N}}$ given by a stream of data $\sigma \in C^{\mathbb{N}}$ and, for any $n \in \mathbb{N}$, a value predicate $P_n \in X \Rightarrow \mathbb{N}_\infty$, determining what the cost of any return value would be if it were yielded after n gives.

Given a computation $t \in TX$, the predicate $\Psi(\mathsf{Tal})(t)$ delivered by the runner computes (1) the number n of gives necessary for reaching its return value $x \in X$ (some number of give responses for each get request made), and (2) the cost associated to x at that point, which is $P_n(x)$. This predicate then yields the sum $n + P_n(x)$ as the total cost of the computation.

7.3 Running with Both a Coalgebra and an Algebra Given

A running perspective is possible also in the situation of merge functors where both a coalgebra and an algebra are given, but nothing too exciting happens. In this case, we concern ourselves only with the final merged algebra as produced by the merge functor. Here are some equivalent definitions of monad algebras.

Proposition 7. *For any object $W \in \mathcal{C}$, the following sets are in bijection:*

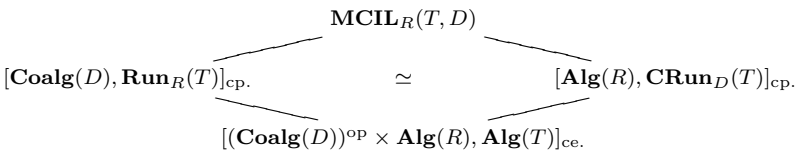
1. *transformations $\mathcal{C}(X, W) \to \mathcal{C}(TX, W)$ natural in X subject to appropriate equations,*[3]
2. *monad morphisms from T to the "external continuation" monad XCnt_W for answer set W,*[4] *defined by $\mathsf{XCnt}_W X := \mathcal{C}(X, W) \pitchfork W$,*
3. *monad algebras of T with carrier W.*

A bijection like that between 2 and 3 holds for T a strong monad when one replaces the external continuation monad XCnt_W with the ordinary continuation monad $\mathsf{Cnt}_W^{\mathsf{Id}}$[5] and monad morphisms with strong monad morphisms [8]. In **Set**, the two continuation monads are isomorphic, every functor is uniquely strong and every natural transformation is strong; therefore, the two bijections become the same.

By Corollary 1, R-residual interaction laws of T, D are in a bijection with functors sending a comonad coalgebra of D with carrier Y and a monad algebra of R with carrier Z to a monad algebra of T with carrier $Y \Rightarrow Z$. By Proposition 7, such algebras are in a bijection with $\mathcal{C}(X \times Y, Z) \to \mathcal{C}(TX \times Y, Z)$ natural in X subject to two equations ("state and continuation based runners"), which amount to natural transformations $TX \times Y \to \mathsf{XCnt}_Z(X \times Y)$ (XCnt_Z-residual state-based runners) or $\mathsf{XCost}_Y(X \Rightarrow Z) \to TX \Rightarrow Z$ (XCost_Y-fuelled continuation-based runners) where $\mathsf{XCost}_Y W = \mathcal{C}(Y, W) \bullet Y$ is the "external costate" monad, with $\bullet$ denoting tensor. They are also in a bijection with monad morphisms to the monad $\mathsf{XCnt}_{Y \Rightarrow Z}$. This monad is isomorphic both to the external-continuation-transformed state monad defined by $\mathsf{XCntSt}_{Y,Z} X := Y \Rightarrow \mathsf{XCnt}_Z(X \times Y)$ and the external-costate-transformed continuation monad defined by $\mathsf{XCostCnt}_{Y,Z} X := \mathsf{XCost}_Y(X \Rightarrow Z) \Rightarrow Z$.

8 Conclusion

We have seen isomorphisms between, among others, the following four descriptions of interactions between a computation and an environment.



where 'cp.' means "carrier-preserving" and 'ce.' means "carrier-exponentiating". The right and bottom corners of the diamond are new. Moreover, just as interaction laws are the same as monad morphisms from T to $D \multimap R$, for each of the

[3] Also known as monad algebras of T with carrier W in "no-iteration" form.
[4] Also called the endomorphism monad.
[5] Also called the double dualization monad.

three types of specializations of interaction laws (based on a coalgebra or/and an algebra), runners of the corresponding type also amount to monad morphisms from T to specific monads. This is also new for the right and bottom corners.

Algebras $\xi : T(Y \Rightarrow Z) \to Y \Rightarrow Z$ delivered by a merge functor do not mention interaction laws or comonads. As such, they are suitable for developments purely in terms of monads and their algebras. If an algebra is a continuous morphism in the category of ω-cpos, and its carrier set forms a complete lattice, then it gives rise to a congruent notion of program equivalence (as seen in previous work [19,23]). It should be relatively easy to extend developed theory to the algebraically compact setting of ω-cpos, using a construction like the one from Sect. 6 to specify R-residual interaction laws between D and T for $TX := \mu V.\,(X + FV)_\perp$. We want to investigate what such a notion of program-environment equivalence would look like.

On the other hand, the merged algebra created using the tools of this paper can be used as a basis for defining and verifying properties of programs. In particular, the emphasis on state predicates makes it perfectly suitable for formulating Hoare logic judgments [6]. Consider a postcondition given by $Q : X \times Y \to Z$, which gives for each possible return value from X and final state from Y a quantitative degree of truth from X. Then, a computation over X, which is an element t of TX, can be transformed using Q into an element of $T(Y \Rightarrow Z)$. Using the merged algebra, we can compute the weakest precondition $\mathsf{wp}(t, Q) : Y \to Z$, associating to each possible initial state the corresponding final degree of truth. In Hoare logic style, we can then formulate that, given a precondition $P : Y \to Z$, $\{P\}\, t\, \{Q\}$ holds if, for all $y \in Y$, $P(y) \leq \mathsf{wp}(t, Q)(y)$ (assuming a partial order on Z). If this is applied to the example of probability with global store, we retrieve the usual notion of probabilistic Hoare logic [11]. More generally, we see this as a potential framework for a flexible Hoare-style logic on (quantitative) state predicates.

Another subject for future research is the *cascading* of interaction laws. If we have two interaction laws, each with their own notion of environment, and the second interacts with the residual effects of the first, we can combine them into one. This way, computations interact with two layers of environment simultaneously. Using the Day convolution to parallel-compose the comonads representing the two notions of environment, we can cascade the interaction laws into a single law. A similar construction can be done on merge functors so that the two constructions correspond.

Acknowledgements. Exequiel Rivas found out and told us that stateful runners have been studied under the name of monad coalgebras.

T.U. was supported by the Icelandic Research Fund project grant no. 196323-052 and by the Estonian Ministry of Education and Research institutional research grant no. IUT33-13. N.V. was supported by the Estonian IT Academy research measure (the European Social Fund project no. 2014-2020.4.05.19-0001).

References

1. Aczel, P., Adámek, J., Milius, S., Velebil, J.: Infinite trees and completely iterative theories: a coalgebraic view. Theor. Comput. Sci. **300**(1–3), 1–45 (2003)
2. Ahman, D., Bauer, A.: Runners in action. In: Müller, P. (ed.) ESOP 2020. LNCS, vol. 12075, pp. 29–55. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-44914-8_2
3. Bauer, A.: What is algebraic about algebraic effects and handlers? arXiv eprint 1807.05923 [cs.LO] (2018). https://arxiv.org/abs/1807.05923
4. Capobianco, S., Uustalu, T.: A categorical outlook on cellular automata. In: Kari, J. (ed.) Proceedings of 2nd Symposium on Cellular Automata, JAC 2010. TUCS Lecture Notes, vol. 13, pp. 88–89. University of Turku, Turku (2010)
5. Hasuo, I.: Generic weakest precondition semantics from monads enriched with order. Theor. Comput. Sci. **604**, 2–29 (2015). https://doi.org/10.1016/j.tcs.2015.03.047
6. Hoare, C.A.R.: An axiomatic basis for computer programming. Commun. ACM **26**(1), 53–56 (1983). https://doi.org/10.1145/357980.358001
7. Katsumata, S., Rivas, E., Uustalu, T.: Interaction laws of monads and comonads. In: Proceedings of 35th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2020, pp. 604–618. ACM, New York (2020). https://doi.org/10.1145/3373718.3394808
8. Kock, J.: On the double dualization monads. Math. Scand. **27**, 151–165 (1970). https://doi.org/10.7146/math.scand.a-10995
9. Møgelberg, R.E., Staton, S.: Linear usage of state. Log. Meth. Comput. Sci. **10**(1), 1–52 (2014). https://doi.org/10.2168/lmcs-10(1:17)2014
10. Moggi, E.: Notions of computation and monads. Inf. Comput. **93**(1), 55–92 (1991). https://doi.org/10.1016/0890-5401(91)90052-4. Article 17
11. Morgan, C., McIver, A., Seidel, K.: Probabilistic predicate transformers. ACM Trans. Program. Lang. Syst. **18**(3), 325–353 (1996). https://doi.org/10.1145/229542.229547
12. Petricek, T., Orchard, D., Mycroft, A.: Coeffects: a calculus of context-dependent computation. SIGPLAN Not. **49**(9), 123–135 (2014). https://doi.org/10.1145/2692915.2628160
13. Plotkin, G., Power, J.: Tensors of comodels and models for operational semantics. Electron. Notes Theor. Comput. Sci. **218**, 295–311 (2008). https://doi.org/10.1016/j.entcs.2008.10.018
14. Plotkin, G., Power, J.: Notions of computation determine monads. In: Nielsen, M., Engberg, U. (eds.) FoSSaCS 2002. LNCS, vol. 2303, pp. 342–356. Springer, Heidelberg (2002). https://doi.org/10.1007/3-540-45931-6_24
15. Plotkin, G.D., Power, J.: Algebraic operations and generic effects. Appl. Categ. Struct. **11**, 69–94 (2003). https://doi.org/10.1023/a:1023064908962
16. Plotkin, G.D., Pretnar, M.: Handling algebraic effects. Log. Meth. Comput. Sci. **9**(4), 1–36 (2013). https://doi.org/10.2168/lmcs-9(4:23)2013. Article 23
17. Poinsot, L., Porst, H.E.: Internal coalgebras in cocomplete categories: generalizing the Eilenberg-Watts theorem. J. Algebra Appl. (to appear). https://doi.org/10.1142/s0219498821501656
18. Power, J., Shkaravska, O.: From comodels to coalgebras: state and arrays. Electron. Notes Theor. Comput. Sci. **106**, 297–314 (2004). https://doi.org/10.1016/j.entcs.2004.02.041

19. Simpson, A., Voorneveld, N.: Behavioural equivalence via modalities for algebraic effects. ACM Trans. Program. Lang. Syst. **42**(1), 1–45 (2020). https://doi.org/10.1145/3363518. Article 4
20. Uustalu, T.: Stateful runners of effectful computations. Electron. Notes Theor. Comput. Sci. **319**, 403–421 (2015). https://doi.org/10.1016/j.entcs.2015.12.024
21. Uustalu, T., Vene, V.: The essence of dataflow programming. In: Horváth, Z. (ed.) CEFP 2005. LNCS, vol. 4164. Springer, Heidelberg (2006). https://doi.org/10.1007/11894100_5
22. Uustalu, T., Vene, V.: Comonadic notions of computation. Electron. Notes Theor. Comput. Sci. **203**(5), 263–284 (2008). https://doi.org/10.1016/j.entcs.2008.05.029
23. Voorneveld, N.: Quantitative logics for equivalence of effectful programs. Electron. Notes Theor. Comput. Sci. **347**, 281–301 (2019). https://doi.org/10.1016/j.entcs.2019.09.015