Available online at www.sciencedirect.com

ScienceDirect

Electronic Notes in Theoretical Computer Science

Electronic Notes in Theoretical Computer Science 319 (2015) 403–421

www.elsevier.com/locate/entcs

Stateful Runners of Effectful Computations

Tarmo Uustalu[1]

Institute of Cybernetics, Tallinn University of Technology,
Akadeemia tee 21, 12618 Tallinn, Estonia

Abstract

What structure is required of a set so that computations in a given notion of computation can be run statefully with this set as the state set? For running nondeterministic computations statefully, a resolver structure is needed; for interactive I/O computations, a "responder-listener" structure is necessary; to be able to serve stateful computations, the set must carry the structure of a lens. We show that, in general, to be a stateful runner of computations for a monad corresponding to a Lawvere theory (defined as a set equipped with a monad morphism between the given monad and the state monad for this set) is the same as to be a comodel of the theory, i.e., a coalgebra of the corresponding comonad. We work out a number of instances of this observation and also compare runners to handlers.

Keywords: effects, monads, Lawvere theories, comodels, state monads, handlers

1 Introduction

This paper is about Moggi's monad-based and Plotkin and Power's Lawvere theories based approaches to effectful computation [8,10].

Given a monad (T, η, μ), a computation of a value in X is an element of $T\,X$. Computations are there to compute values, so we consider it natural to wish to extract these values, to *run* computations. Ideally, we might want to have at our disposal a polymorphic function $\theta : \forall X.\, T\,X \to X$ for extracting values from computations, but this is generally too much to ask (although it is possible, e.g., for writer monads).

However we can often produce a value, if we are allowed to rely on some input—think of it as an initial state—drawn from some set C with suitable structure. For example, if we have a finitely nondeterministic computation in the sense of a binary wellfounded leaf tree, a bitstream can be used to identify a leaf. As running should reasonably be compositional in the sense that running the sequence of two computations should be the same as composing two runs, a run should not only

[1] tarmo@cs.ioc.ee

http://dx.doi.org/10.1016/j.entcs.2015.12.024

depend on an initial state, but also return a final state (that can serve as the initial state for another run). In the case of nondeterminism and bitstreams, the final state could be the remainder of the bitstream provided as the initial state. So in general we might want to look for a polymorphic function $\theta : \forall X.\, T\, X \to T^C\, X$ where (T^C, η^C, μ^C) is the state monad for C as the state set. The compositionality we want amounts to θ being not just a natural transformation, but a monad morphism.

In this paper, we answer the question of when a set C can be used to run computations in a monad (T, η, μ) statefully, assuming that the monad corresponds to a Lawvere theory. The answer is: C has to carry a comodel of the Lawvere theory (i.e., a coalgebra of the corresponding comonad). We spell out a number of instances of this generality, for nondeterminism, interactive I/O and stateful computations. This is an easy exercise, but the results are quite instructive, we find. For some versions of nondeterminism, for instance, runners can only recover a part of the information in a given computation; other versions of nondeterminism admit only trivial runners that reveal nothing about the computation. So some variations of nondeterminism are inherently more operational than others.

Runners are somewhat similar to handlers, but one bigger difference is that runners are polymorphic in the value set. For example, handling allows one to extract a value from a nondeterministic computation (a binary wellfounded leaf tree) over a specific value set that carries a binary operation by folding this operation over the leaf labels. (If for us a nondeterministic computation is a nonempty list of values, this operation must be associative.) Running does not allow such things. In our view, the pragmatics of handlers and runners are different: handlers are a programming language construct, but runners are compilation schemes.

The paper is organized as follows. In Section 2, we review the few basic facts about Lawvere theories, models and comodels that we need. In Section 3, we show that stateful runners for a monad corresponding to a Lawvere theory are in a bijection with comodels of the theory (coalgebras of the corresponding comonad). We also compare this observation to a fact about monad morphisms to continuation monads—a different type of runners. In Section 4, we work out the instances for nondeterminism, interactive I/O and stateful computation. Just before concluding, in Section 5, we compare runners to handlers.

2 Lawvere theories, models, comodels

We begin by reviewing the most basic definitions and facts about finitary Lawvere theories and models (for a proper exposition, see, e.g., [6]) as well as Power's comodels [13,11]. Countable Lawvere theories and κ-ary Lawvere theories for a regular cardinal κ are defined analogously.

Lawvere theories

A *(finitary) Lawvere theory* is given by a small category $\mathbb{L}$ with finite products and a functor $L : \mathbb{F}^{\mathrm{op}} \to \mathbb{L}$ that is identity on objects and strictly preserves the finite products of $\mathbb{F}^{\mathrm{op}}$.

Here $\mathbb{F}$ is the category of finite cardinals, i.e., the skeleton of the category of finite sets. It is a strict monoidal category wrt. finite coproducts, in fact it is the free such category on the one-object category.

A theory can (non-uniquely) be specified by a presentation, i.e., by some subset of the maps $\textsc{op}_j : I_j \to O_j$ of $\mathbb{L}$ (*operations*) from which all other maps are definable together with some subset of the commuting diagrams $\textsc{lhs}_k = \textsc{rhs}_k$ of $\mathbb{L}$ (*equations*) from which all other commuting diagrams follow.

Notice that one can always do with operations of arities $I \to 1$ only. As $O = \coprod_{o \in O} 1$, any operation $\textsc{op} : I \to O$ can be replaced with O many operations $\textsc{op}^o : I \to 1$ via $\textsc{op}^o = \mathsf{in}_o \circ \textsc{op}$ and $\textsc{op} = (\coprod_{o \in O} \textsc{op}^o) \circ \nabla$.[2])

Models

A *model* of a theory $(\mathbb{L}, L)$ is given by a functor $[\![-]\!] : \mathbb{L} \to \mathbf{Set}$ that preserves the finite products of $\mathbb{L}$ (non-strictly).

To give a model of $(\mathbb{L}, L)$, it suffices to specify a set

$$A = [\![1]\!]$$

since, for any other object Y, we have $[\![Y]\!] = [\![\coprod_{y \in Y} 1]\!] \cong \prod_{y \in Y} [\![1]\!] = [\![1]\!] \Leftarrow Y$[3] together with functions

$$op_j = \Lambda^{-1} [\![\textsc{op}_j]\!] : O_j \times ([\![1]\!] \Leftarrow I_j) \to [\![1]\!]$$

since, for any other map f, $[\![f]\!]$ is then uniquely determined by functoriality and preservation of finite products. Moreover, any set A with functions $op_j : O_j \times (A \Leftarrow I_j) \to A$ defines a model, provided that the equations $lhs_k = rhs_k$ hold for the derived functions lhs_k, rhs_k.

Theories and monads

A theory defines a unique monad whose algebras are essentially the same as its models.

The monad corresponding to a theory $(\mathbb{L}, L)$ is a quotient of a free monad. The underlying functor is

$$T\,X = T_0\,X / {\sim}^*_X$$

where $T_0\,X$ is the set defined inductively by

$$\frac{x : X}{\mathsf{var}\,x : T_0\,X} \qquad \frac{o : O_j \quad f : T_0\,X \Leftarrow I_j}{\mathsf{op}_j\,(o, f) : T_0\,X}$$

(so that, in a more compact notation, $T_0\,X = \mu Z.\, X + \coprod_j O_j \times (Z \Leftarrow I_j)$) and $\sim_X$

[2] We write maps of $\mathbb{L}$ in terms of the operations of the presentation, maps of $\mathbb{F}$, the product bifunctor of $\mathbb{L}$, which we denote $+$ (sic!) to agree with the notation for the coproducts of $\mathbb{F}$, and composition of $\mathbb{L}$. Note that maps of $\mathbb{F}$ have their directions reversed in $\mathbb{L}$, so $\mathsf{in}_o : 1 \to O$ in $\mathbb{F}$, but $\mathsf{in}_o : O \to 1$ in $\mathbb{L}$ etc.

[3] To avoid the need to explicitly use the symmetry of $\times$ in the examples, we will use two exponential functors $\Leftarrow$ and $\Rightarrow$; think of $- \Leftarrow Y$ as the right adjoint of $Y \times -$ and $Y \Rightarrow -$ as the right adjoint of $- \times Y$.

is a binary relation on $T_0\,X$ defined inductively by

$$\frac{\forall i : I_i.\, f\, i \sim_X f'\, i}{\mathsf{op}_j\,(o, f) \sim_X \mathsf{op}_j\,(o, f')} \quad \frac{}{\mathsf{lhs}_k\,(o, f) \sim_X \mathsf{rhs}_k\,(o, f)}$$

The unit η is var and the multiplication μ is defined recursively by $\mu\,(\mathsf{var}\,t) = t$ and $\mu\,(\mathsf{op}_j\,(o, f)) = \mathsf{op}_j(o, \lambda i.\, \mu\,(f\, i))$.

Algebras of (T, η, μ) are essentially the same as models. A model $(A, (op_j : O_j \times (A \Leftarrow I_j) \to A)_j)$ and an algebra $(A, \alpha : T\,A \to A)$ are interdefinable recursively by $\alpha\,(\mathsf{var}\,x) = x$ and $\alpha\,(\mathsf{op}_j\,(o, f)) = op_j(o, \lambda i.\, \alpha\,(f\, i))$ and by $op_j\,(o, f) = \alpha\,(\mathsf{op}_j\,(o, \lambda i.\, \mathsf{var}\,(f\, i)))$.

The monad corresponding to a theory is finitary. And any monad corresponds to at most one theory in this fashion.

Comodels

We are now prepared to discuss Power's notion of comodels.

A *comodel* of a theory $(\mathbb{L}, L : \mathbb{F}^{\mathrm{op}} \to \mathbb{L})$ is given by a functor $\langle\!\langle - \rangle\!\rangle : \mathbb{L}^{\mathrm{op}} \to \mathbf{Set}$ that preserves the finite coproducts of $\mathbb{L}^{\mathrm{op}}$ (recall that a model was a functor from $\mathbb{L}$ preserving its finite products).

To give a comodel, it suffices to specify a set

$$C = \langle\!\langle 1 \rangle\!\rangle$$

since $\langle\!\langle X \rangle\!\rangle = \langle\!\langle \coprod_{x \in X} 1 \rangle\!\rangle \cong \coprod_{x \in X} \langle\!\langle 1 \rangle\!\rangle = \langle\!\langle 1 \rangle\!\rangle \times X$, together with functions

$$\overline{op}_j = \langle\!\langle \mathrm{OP}_j \rangle\!\rangle : \langle\!\langle 1 \rangle\!\rangle \times O_j \to \langle\!\langle 1 \rangle\!\rangle \times I_j$$

where we often split $\overline{op}_j$ as $\langle \overline{opn}_j, \overline{ops}_j \rangle$ (with n and s mnemonic for "next" and "show"). Also, any set C with functions $\overline{op}_j : C \times O_j \to C \times I_j$ defines a comodel, provided that the equations $\overline{lhs}_k = \overline{rhs}_k$ hold for the derived functions $\overline{lhs}_k$, $\overline{rhs}_k$.

Theories and comonads

Besides defining a (finitary) monad whose algebras are the same as models, a theory also defines a comonad with the property that comodels of the theory are the same as coalgebras of the comonad.

The comonad corresponding to a theory $(\mathbb{L}, L)$ is a subcomonad of a cofree comonad. The underlying functor is defined by

$$D\,X = D_0\,X \mid ok_X$$

where $D_0\ X$ is a set defined coinductively by [4]

$$\frac{d : D_0\,X}{\overline{\mathsf{var}}\,d : X} \quad \frac{d : D_0\,X \quad o : O_j}{\overline{\mathsf{opn}}_j\,(d, o) : D_0\,X} \quad \frac{d : D_0\,X \quad o : O_j}{\overline{\mathsf{ops}}_j\,(d, o) : I_j}$$

[4] We write coinductive definitions in a destructor-based fashion (as opposed to the constructor-based style commonly used in proof assistants), as this works more smoothly.

(so that, in a more compact notation, $D_0\,X \cong \nu Z.\, X \times \prod_j (O_j \Rightarrow Z \times I_j)$) and ok_X is a predicate on $D_0\,X$ defined coinductively by

$$\frac{ok_X\,d}{ok_X\,(\overline{\mathsf{opn}}_j\,(d,o))} \qquad \frac{ok_X\,d}{\overline{\mathsf{lhsn}}_k(d,o) = \overline{\mathsf{rhsn}}_k(d,o)} \qquad \frac{ok_X\,d}{\overline{\mathsf{lhss}}_k(d,o) = \overline{\mathsf{rhss}}_k(d,o)}$$

The counit ε is $\overline{\mathsf{var}}$. The comultiplication δ is defined corecursively by $\overline{\mathsf{var}}\,(\delta\,d) = d$, $\overline{\mathsf{opn}}_j\,(\delta\,d,o) = \delta\,(\overline{opn}_j\,(d,o))$, $\overline{\mathsf{ops}}_j\,(\delta\,d,o) = \overline{ops}_j\,(d,o)$.

A comodel $(C, \overline{op}_j : C \times O_j \to C \times I_j)_j)$ is interdefinable with a coalgebra $(C, \gamma : C \to D\,C)$ corecursively by $\overline{\mathsf{var}}\,(\gamma\,c) = c$, $\overline{\mathsf{opn}}_j\,(\gamma\,c,o) = \gamma\,(\overline{opn}_j\,(c,o))$, $\overline{\mathsf{ops}}_j\,(\gamma\,c,o) = \overline{ops}_j\,(c,o)$, and by $\overline{opn}_j\,(c,o) = \overline{\mathsf{var}}\,(\overline{\mathsf{opn}}_j\,(\gamma\,c,o))$, $\overline{ops}_j\,(c,o) = \overline{\mathsf{ops}}_j\,(\gamma\,c,o)$.

The comonad corresponding to a theory in the above fashion is in general non-finitary. Also, one comonad can correspond to many theories. (E.g., all theories with at least one nullary operation ($\textsc{op} : 0 \to O$ with $O \neq 0$) have the initial comonad ($D\,X = 0$) as the corresponding comonad.)

3 Stateful runners = comodels

We are prepared to prove the theorem that this paper revolves around. We prove that a stateful runner is the same as a comodel.

Proposition 3.1 *Given a Lawvere theory $(\mathbb{L}, L)$, let (T, η, μ) be the monad corresponding to the theory. Given a set C, let (T^C, η^C, μ^C) be the state monad for C. There is a bijection between monad morphisms from (T, η, μ) to (T^C, η^C, μ^C) and comodels on C (i.e., coalgebras of the comonad (D, ε, δ) corresponding to $(\mathbb{L}, L)$).*

Proof. Let the Lawvere theory $(\mathbb{L}, L)$ be given by operations $\textsc{op}_j : I_j \to O_j$ and equations $\textsc{lhs}_k = \textsc{rhs}_k$. The corresponding monad (T, η, μ) is then constructed as described in the previous section.

Given a comodel $(\overline{op}_j : C \times O_j \to C \times I_j)_j$, the monad morphism $\theta : \forall X.\, T\,X \to (C \times X) \Leftarrow C$ is defined by recursion on $t : T_0\,X$ by

$$\begin{aligned} \theta_X\,(\mathsf{var}\,x) &= \lambda c.\,(c, x) \\ \theta_X\,(\mathsf{op}_j\,(o, f)) &= \lambda c.\,\theta_X\,(f\,(\overline{ops}_j\,(c,o))\,(\overline{opn}_j\,(c,o)) \end{aligned}$$

For this definition to be legitimate, θ_X must send $\sim_X$-related computations in $T_0\,X$ to equal computations in $T^C\,X$. This is proved by induction on the proof of $t \sim_X t'$ from $\overline{lhsn}_k = \overline{rhsn}_k$ and $\overline{lhss}_k = \overline{rhss}_k$.

The unit preservation law of a monad morphism holds trivially. The multiplication preservation law is a "substitution lemma" that is proved by induction on $t : T_0\,(T_0\,X)$.

In the converse direction, given a monad morphism θ, we define the comodel $(\overline{op}_j)_j$ by

$$\overline{op}_j\,(c,o) = \theta_{I_j}(\mathsf{op}_j\,(o, \lambda i.\,\mathsf{var}\,i))\,c$$

The equations $\overline{lhsn}_k = \overline{rhsn}_k$ and $\overline{lhss}_k = \overline{rhss}_k$ follow from θ_X sending $\sim_X$-related computations in $T_0\,X$ to equal computations in $T^C\,X$.

The roundtrip from a comodel to a monad morphism and back is straightforwardly identity. The other roundtrip is identity thanks to naturality of θ. □

We compare this theorem to the following well known theorem (see, e.g., [5]) about running with continuations.

Proposition 3.2 *Given a monad (T, η, μ) and a set A. Let (T^A, η^A, μ^A) be the continuation monad for A. Monad morphisms θ from (T, η, μ) to (T^A, η^A, μ^A) are in a bijection with (T, η, μ)-algebra structures α on A.*

Proof. Given an algebra structure $\alpha : T\,A \to A$, the monad morphism $\theta : \forall X.\, T\,X \to (A \Leftarrow X) \Rightarrow A$ is defined by $\theta_X\,t = \lambda f.\, \alpha\,(T\,f\,t)$; the laws of a monad morphism follow from the laws of an algebra.

In the converse direction, given a monad morphism θ, the corresponding algebra structure α is defined by $\alpha\,t = \theta_A\,t\,\mathsf{id}_A$; the laws of an algebra follow from the laws of a monad morphism.

The roundtrip from an algebra to a monad morphism and back is straightforwardly identity. The proof of the other roundtrip being identity relies on the naturality of θ. □

Notice that in the first theorem we use a Lawvere theory to relate a monad and a comonad (however the Lawvere theory is determined by the monad). The second theorem can be stated without referring to a Lawvere theory.

A more important difference is this. Given any monad, for any value set X, one can find a runner with continuations (A, θ) such that θ_X is mono, i.e., all information about a computation over a given value set X in the given monad is retained in its counterpart in the continuation monad. Indeed, for any X, invoking the free algebra $(T\,X, \mu_X)$, we have $\theta_X\,t\,\eta_X = t$.

In the case of stateful running, it is much more difficult to achieve θ_X being mono. As we will see shortly, it is easy to construct examples where no state set C is sufficient to recover computations over a given value set X.

Both theorems can be strengthened to isomorphisms of categories; we skip the details here.

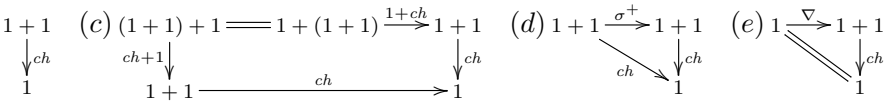
4 Instances

4.1 Nondeterminism

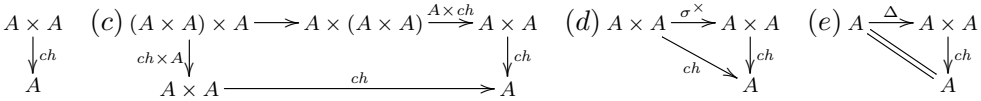
Let us see the above proposition at work on the example of various finite nondeterminism monads as well as the partiality monad.

Finite nondeterminism

First we consider theories given by the following operation and some of the following equations:



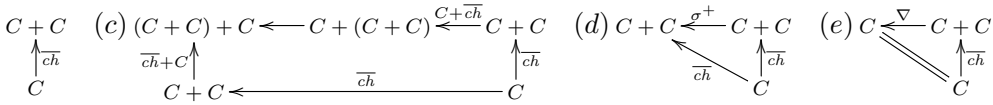
A model of each such theory is given by a set A carrying the following function and satisfying the correct equations from among the following:



We see that models of these theories are exactly what we expect: without any equations we get magmas, with (c) semigroups, with (c), (d) commutative semigroups, with (c), (d), (e) semilattices, with (d) alone commutative magmas.

The corresponding monads are those of binary leaf trees (free magmas, $T\,X \cong \mu Z.\, X + Z \times Z$), nonempty lists (free semigroups), nonempty finite multisets (free commutative semigroups), nonempty finite sets (free semilattices), binary unordered leaf trees (free commutative magmas). All of them are quotients of the first monad. They all model nondeterminism to some level of granularity; which monad to use depends on what exactly one wants to track as the nondeterminism effect (and in fact on whether one wants to be able to resolve nondeterminism, as we will see shortly). The theory view of these monads tells us that the single operation *ch* is complete for programming finitely nondeterministic functions.

A comodel (a runner for nondeterministic computations) is a set C with the following function satisfying the intended equations from among the following:



Reformulating minimally, a comodel is a set C with a function $\overline{ch} = \langle \overline{chn}, \overline{chs} \rangle : C \to C \times 2$ satisfying the appropriate equations. For the theory without equations, it is nothing but a resolver (scheduler) for the finest notion of nondeterminism that remembers the order that binary choices are made, the order of options in binary choices etc. It is a machine that can make a binary choice on request and go to a new state. Given a nondeterministic computation, which is binary leaf-labelled tree in this case, it can thus choose a path from the root to some leaf.

Equation (e) forces that $\overline{chn}\, x = x$. Equation (c) requires $\overline{chs}\, (\overline{chn}\, x) = \overline{chs}\, x$ and $\overline{chn}\, (\overline{chn}\, x) = \overline{chn}\, x$, which are trivially fulfilled when $\overline{chn}$ is identity. So a resolver for the version of nondeterminism where order of binary choices is considered irrelevant and a binary choice between two equal options is considered the same as no choice at all (so computations are nonempty square-free lists of values, i.e., lists

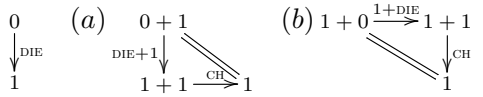
with no sublist occurring twice in a row) amounts to a set C with an unconstrained function $\overline{chs} : C \to 2$, This is a machine that always makes its choice without changing its state, so, given a leaf tree (identified with other leaf trees that flatten into the same nonempty list), it walks down from the root by always turning to the left or always turning to the right, eventually reaching the leftmost or rightmost leaf (the first or last position of the list). Note that the other leaves (inner positions of the list) are unreachable for a runner—they are not addressable "crisply" enough. [5]

In a comodel for any theory containing equation (d), it must be that $\mathsf{not}\,(\overline{chs}\,x) = \overline{chs}\,x$, which can only hold when $C \cong 0$. This says that, as soon as the order of the options in a choice is considered immaterial, resolving nondeterminism is impossible, apart from the uninteresting degenerate case.

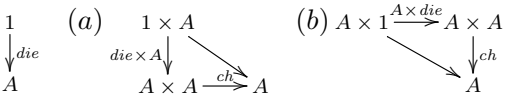
The comonad for the equationless theory is that of streams of states and bits, $D\,X \cong \nu Z.\, X \times (2 \times Z)$. The comonads for more specific theories are subcomonads. In particular, the comonad for the theory with (c) alone has $D\,X \cong X \times (2 \times X)$, the theory with both (c) and (e) has $D\,X \cong X \times 2$. The comonads for theories containing (d) have $D\,X \cong 0$.

Finite nondeterminism with failure

Let us consider extending the theories considered with the following operation and, possibly, the following equations:



A model is now a set A supporting the following function satisfying the intended equations from among following:

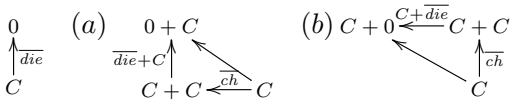


Models are pointed magmas, monoids, commutative semilattices with a bottom, commutative pointed magmas. The monads are the monad of nullary-binary leaf trees (free pointed magmas, $T\,X \cong \mu Z.\, X + 1 + Z \times Z$) and its different quotients—the monads of lists (free monoids), finite multisets (free commutative monoids), finite sets (free commutative semilattices with bottom), nullary-binary unordered leaf trees (free pointed commutative magmas). They all model notions of nondeterminism where also "no-option" choices are allowed.

A comodel is a set C with the following function satisfying the intended ones of

[5] These two elements of $T\,3$ are equal: $t_0 = \mathsf{ch}\,(\mathsf{ch}\,(\mathsf{var}\,0, \mathsf{var}\,1), \mathsf{var}\,2)$ and $t_1 = \mathsf{ch}\,(\mathsf{var}\,0, \mathsf{ch}\,(\mathsf{var}\,1, \mathsf{var}\,2))$. But $t_0 = \mu\,t'_0$ and $t_1 = \mu\ t'_1$ for $t'_0 = \mathsf{ch}\,(\mathsf{var}(\mathsf{ch}\,(\mathsf{var}\,0, \mathsf{var}\,1)), \mathsf{var}\,(\mathsf{var}\,2))$ and $t'_1 = \mathsf{ch}\,(\mathsf{var}\,(\mathsf{var}\,0), \mathsf{var}\,(\mathsf{ch}\,(\mathsf{var}\,1, \mathsf{var}\,2)))$ in $T\,(T\,3)$. Both t'_0 and t'_1 are of the form $\mathsf{ch}\,(\mathsf{var}(\ldots), \mathsf{var}\,(\ldots))$. A runner that is able to extract from t_0 the value 1, must process t'_0 by first going to the left, but then it must do the same to t'_1 in which case it cannot extract the value 1 from t_1, which was equal to t_0.

the following equations:



It is immediate that there are no interesting comodels: the carrier of a comodel must be empty even in the case of no equations. The comonads are all constant 0 ($D\,X \cong 0$). It is impossible (except for the uninteresting degenerate case of an impossible initial state) to resolve a nondeterministic computation that may fail.

Observe that the same happens with any theory with one or more nullary operations (OP : $0 \to O$ where $O \neq 0$) or, in terms of monads, with any monad such that $T\,0 \neq 0$.

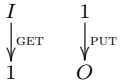
Partiality

Finally we could also skip CH and consider the theory with just DIE and no equations.

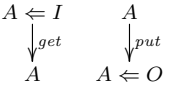
Models of this theory are pointed sets. The monad is the maybe monad (of sets with an added point, free pointed sets, $T\,X \cong 1 + X$), commonly used for modelling partiality. We learn the unsurprising fact that DIE is the sole operation needed for programming partial functions. We know already that there are no interesting comodels.

4.2 Interactive input/output

We move on to consider examples of other types. For the start, take two sets I, O, and consider the very simple theory with the following two operations and no equations:

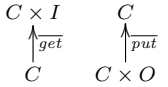


A model is a set A endowed with functions



The monad is the free monad defined by $T\,X = \mu Z.\, X + (Z \Leftarrow I) + O \times Z$. We recognize in it the monad for interactive input/output with I and O as the input and output alphabets.

A comodel is a set C with two functions



It is a runner for interactive input/output, a machine that can provide input and consume output, changing its state. The comonad is the cofree comonad defined by $D\,X = \nu Z.\, X \times (Z \times I) \times (O \Rightarrow Z)$.

The case of interactive input only is covered by the special case $O = 0$ when we could just as well drop the operation PUT as forced and void of information.

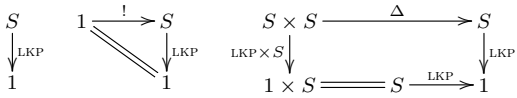
Allowing interactive output only corresponds to dropping the GET operation. This is different from the case $I = 0$ (input from an empty alphabet, leads to partiality), as well as from the case $I = 1$ (input from a singleton alphabet). But via $P = O^*$ (the free monoid on O) it is an instance of writing considered in the next section.

4.3 Stateful computation

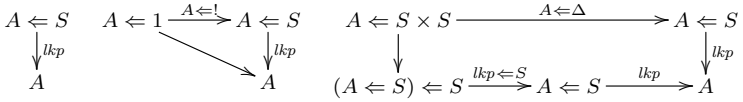
We proceed to stateful computation. We first look at reading only and writing only, to then continue with reading and overwriting (modelled by state monads). We finish by analyzing reading and general updating (modelled by what we call update monads).

Reading

We begin with reading. Take a set S (of states). We look at this theory:



A model is a set A with



By the general construction, the monad for this theory is given by $T\,X = T_0\,X/{\sim_X^*}$ with $T_0\,X$ and $\sim_X$ defined inductively by

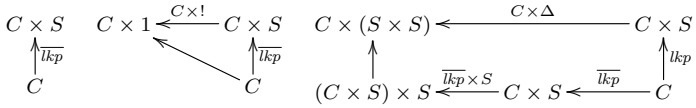
$$\frac{x : X}{\mathsf{var}\,x : T_0\,X} \qquad \frac{f : T_0\,X \Leftarrow S}{\mathsf{lkp}\,f : T_0\,X}$$

(so that $T_0\,X \cong \mu Z.\ X + (Z \Leftarrow S)$) and

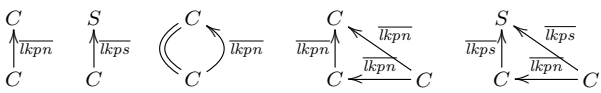
$$\frac{\forall s.\ f\,s \sim_X f'\,s}{\mathsf{lkp}\,f \sim_X \mathsf{lkp}\,f'} \qquad \frac{}{c \sim_X \mathsf{lkp}\,(\lambda s.\ c)} \qquad \frac{}{\mathsf{lkp}\,(\lambda s'.\ \mathsf{lkp}\,(\lambda s.\ f\,s\,s'))\sim_X \mathsf{lkp}\,(\lambda s.\ f\,s\,s)}$$

It is easy to verify that every element of $T\,X$ can be presented in the normal form $\mathsf{lkp}\,(\lambda s.\ \mathsf{var}\,(f\,s))$ for a unique $f : X \Leftarrow S$. It follows that the monad can alternatively be defined without quotienting by $T\,X = X \Leftarrow S$ and $\eta\,x = \lambda s.\ x$, $\mu\,f = \lambda s.\ f\,s\,s$. This is the reader monad for S as the state set.

A comodel is a set C with



or equivalently with

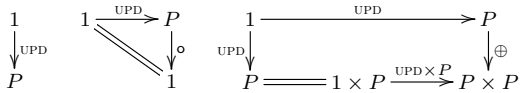


In the latter description, the 1st equation explicitly asks that $\overline{lkpn} = \mathsf{id}_C$, making $\overline{lkpn}$ redundant, and the 2nd and 3rd follow, so we are left with a function $\overline{lkps} : C \to S$ and no equations. A runner for reading amounts thus to a machine that is happy to serve a lookup request with an external state (drawn from set S) and continue then in the same internal state (from set C)—so next time it will provide the same external state again.

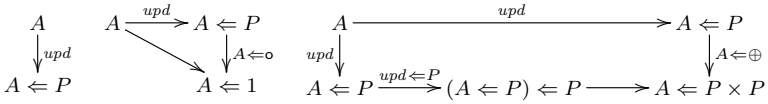
It follows that the comonad can be defined by $D\,X = X \times S$ and $\varepsilon\,(x, s) = x$, $\delta\,(x, s) = ((x, s), s)$. This is the cofree comonad on the constant S functor.

Writing

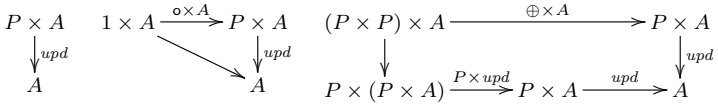
For writing, we take a monoid $(P, \mathsf{o}, \oplus)$ (of updates) and consider the following theory:



A model of this theory is a set A with



or, alternatively, in uncurried form,



which is exactly what it means to be a left action of $(P, \mathsf{o}, \oplus)$.

The general construction tells us that the corresponding monad is given by $T\,X = T_0\,X/{\sim_X^*}$ where $T_0\,X$ and $\sim_X$ are defined inductively by

$$\frac{x : X}{\mathsf{var}\,x : T_0\,X} \qquad \frac{p : P \quad c : T_0\,X}{\mathsf{upd}\,(p, c) : T_0\,X}$$

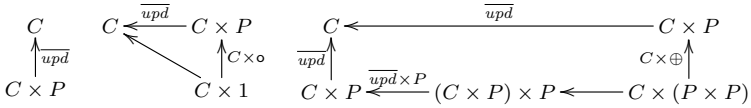
(so that $T_0 X = \mu Z.\, X + P \times Z$) and

$$\frac{c \sim_X c'}{\mathsf{upd}\,(p, c) \sim_X \mathsf{upd}\,(p, c')} \qquad \overline{c \sim_X \mathsf{upd}\,(\mathsf{o}, c)} \qquad \overline{\mathsf{upd}\,(p, \mathsf{upd}\,(p', c)) \sim_X \mathsf{upd}(p \oplus p', c)}$$

We can witness that every element of $T\,X$ can be cast in the form $\mathsf{upd}\,(p, \mathsf{var}\,x)$ for a unique pair $(p, x) : P \times X$. As a consequence, the monad is alternatively definable

by $T\,X = P \times X$ and $\eta\,x = (\mathsf{o}, x)$, $\mu\,(p, (p', x)) = (p \oplus p', x)$. This is the familiar writer monad for $(P, \mathsf{o}, \oplus)$ as the monoid of updates.

A comodel for the theory of writing (a runner for writing computations) is a set C with



i.e., a right action of the monoid. We think of it as a machine that listens to updates and changes its state.

The comonad is constructed by taking $D\,X = D_0\,X \mid ok_X$ where $D_0\,X$ and ok_X are defined coinductively by

$$\frac{c : D_0\,X}{\overline{\mathsf{var}}\,c : X} \qquad \frac{c : D_0\,X \quad p : P}{\overline{\mathsf{upd}}\,(c, p) : D_0\,X}$$

(so that $D_0\,X = \nu Z.\, X \times (P \Rightarrow Z)$) and

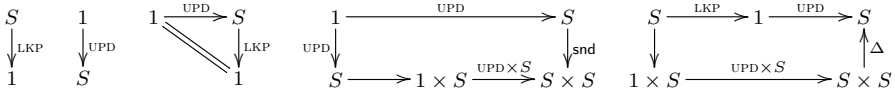
$$\frac{ok_X\,c}{ok_X\,(\overline{\mathsf{upd}}\,(c, p))} \qquad \frac{ok_X\,c}{c = \overline{\mathsf{upd}}\,(c, \mathsf{o})} \qquad \frac{ok_X\,c}{\overline{\mathsf{upd}}\,(\overline{\mathsf{upd}}\,(c, p), p') = \overline{\mathsf{upd}}\,(c, p \oplus p')}$$

Here it is the case that everything that can be learned at all about an element $[\,]$ of $D\,X$ (i.e., an ok element of $D_0\,X$) is summarized in the function $\lambda p.\, \overline{\mathsf{var}}\,(\overline{\mathsf{upd}}\,([\,], p)) : P \Rightarrow X$. It is a universal way of observing elements of $D\,X$ in the sense that, for any set Y, any function $f : D\,X \to Y$ is expressible as $\lambda c.\, g\,(\lambda p.\, \overline{\mathsf{var}}\,(\overline{\mathsf{upd}}\,(c, p)))$ for a unique $g : (P \Rightarrow X) \to Y$. Therefore, the comonad is more succinctly (without carving a subset) defined by $D\,X = P \Rightarrow X$, $\varepsilon\,v = v\,\mathsf{o}$, $\delta\,v = \lambda p.\, \lambda p'.\, v\,(p \oplus p')$.

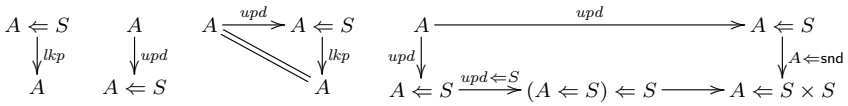
Reading and overwriting

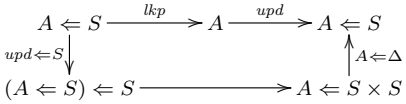
We can now proceed reading and overwriting a state.

Given a set S (of states), the theory of reading and overwriting is given by two operations LKP and UPD

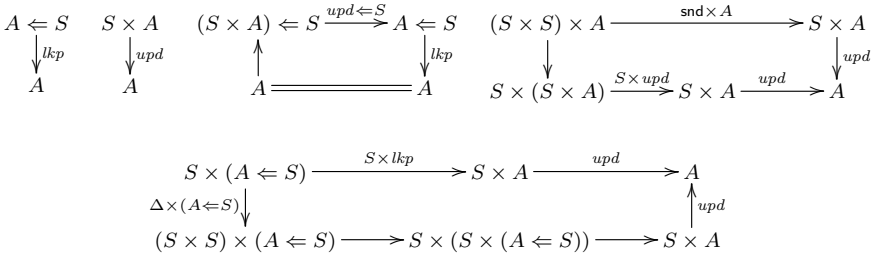


A model is a set A with functions *lkp* and *upd* such that





or, alternatively, in uncurried form,



The corresponding monad is $T\,X = T_0\,X/{\sim_X^*}$ where $T_0\,X$ and $\sim_X$ are defined inductively by

$$\frac{x : X}{\mathsf{var}\,x : T_0\,X} \quad \frac{f : T_0\,X \Leftarrow S}{\mathsf{lkp}\,f : T_0\,X} \quad \frac{s : S \quad c : T_0\,X}{\mathsf{upd}\,(s,c) : T_0\,X}$$

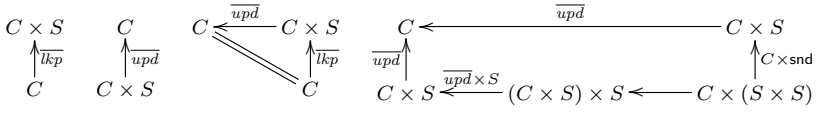
(so that $T_0\,X = \mu Z.\, X + (Z \Leftarrow S) + S \times Z$) and

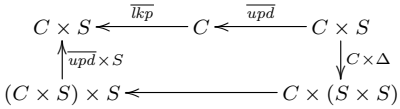
$$\frac{\forall s.\, f\,s \sim_X f'\,s}{\mathsf{lkp}\,f \sim_X \mathsf{lkp}\,f'} \quad \frac{c \sim_X c'}{\mathsf{upd}\,(s,c) \sim_X \mathsf{upd}\,(s,c')}$$

$$\overline{c \sim_X \mathsf{lkp}\,(\lambda s.\, \mathsf{upd}\,(s,c))} \quad \overline{\mathsf{upd}\,(s,\mathsf{upd}\,(s',c)) \sim_X \mathsf{upd}\,(s',c)} \quad \overline{\mathsf{upd}\,(s,\mathsf{lkp}\,f) \sim_X \mathsf{upd}\,(s,f\,s)}$$

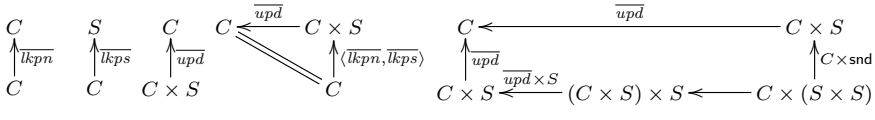
As every element of $T\,X$ can be uniquely presented in the normal form $\mathsf{lkp}\,(\lambda s.\, \mathsf{upd}\,(g\,s, \mathsf{var}\,(h\,s))$ for some $\langle g, h\rangle : (S \times X) \Leftarrow S$, we have that $T\,X \cong (S \times X) \Leftarrow S$ whereby $\eta\,x = \lambda s.\, (s,x)$, $\mu\,f = \lambda s.\, \mathsf{let}\ (s',g) \leftarrow f\,s\ \mathsf{in}\ g\,s'$ —the state monad for S.

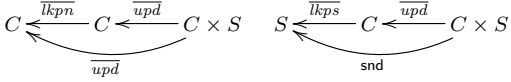
A comodel is a set C together with functions $\overline{lkp}$ and $\overline{upd}$ such that





or, splitting $\overline{lkp} = \langle \overline{lkpn}, \overline{lkps}\rangle$,





Here the 1st and 3rd equation together give that $\overline{lkpn} = \mathsf{id}_C$ making $\overline{lkpn}$ redundant. The 1st equation then simplifies to $\overline{upd} \circ \langle \mathsf{id}_C, \overline{lkps}\rangle = \mathsf{id}_C$ and the 3rd equation becomes tautological. We see that a runner for a stateful computation is a machine

responding to lookups (without changing its state) and listening to overwrites.[6] This structure is known in bidirectional transformations [4] as a *lens*[7] between C and S.[8]

The comonad is $D\,X = D_0\,X \mid ok_X$ where $D_0\,X$ and ok_X are defined coinductively by

$$\frac{c : D_0\,X}{\overline{\mathsf{var}}\,c : X} \quad \frac{c : D_0\,X}{\overline{\mathsf{lkps}}\,c : S} \quad \frac{c : D_0\,X \quad s : S}{\overline{\mathsf{upd}}\,(c,s) : D_0\,X}$$

(so that $D_0\,X = \nu Z.\, X \times S \times (S \Rightarrow Z)$) and

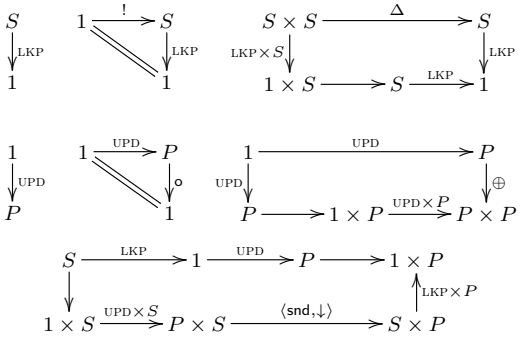
$$\frac{ok_X\,c}{ok_X\,(\overline{\mathsf{upd}}\,(c,s))} \quad \frac{ok_X\,c}{c = \overline{\mathsf{upd}}\,(c, \overline{\mathsf{lkps}}\,c)} \quad \frac{ok_X\,c}{\overline{\mathsf{upd}}\,(\overline{\mathsf{upd}}\,(c,s),s') = \overline{\mathsf{upd}}\,(c,s')} \quad \frac{ok_X\,c}{\overline{\mathsf{lkps}}\,(\overline{\mathsf{upd}}\,(c,s)) = s}$$

All that can be known about an element $[\,]$ of $D\,X$ can be summarized in the universal observation $(\overline{\mathsf{lkps}}\,[\,], \lambda s.\, \overline{\mathsf{var}}\,(\overline{\mathsf{upd}}\,([\,],s))) : S \times (S \Rightarrow X)$. Hence the comonad can also be defined by $D\,X = S \times (S \Rightarrow X)$ and $\varepsilon\,(s,v) = v\,s$ and $\delta\,(s,v) = (s, \lambda s'.\,(s',v))$. This comonad is known as the costate (or array) comonad [13,9].

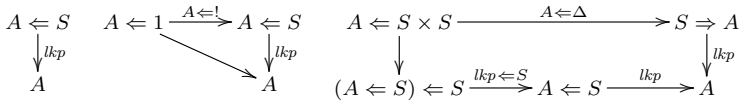
Reading and general writing

Let us also consider reading and general writing (as opposed to just overwriting) in combination.

Given a set S (of states), a monoid $(P, \mathsf{o}, \oplus)$ (of updates) and a right action $\downarrow : S \times P \to S$ (describing application of updates to states), we are interested in the theory given by the following operations and equations:[9]



A model a set A with functions *lkp* and *upd* such that

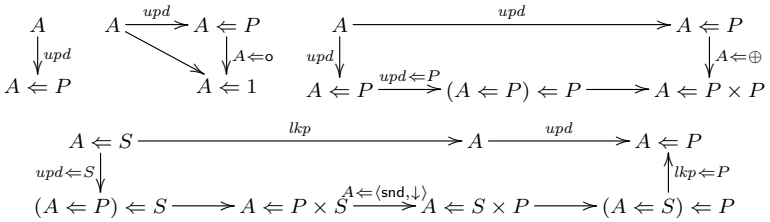


[6] The computation has S as its state set; the runner's state set is C. The final comodel has $C = S$, $\overline{lkps} = \mathsf{id}_S$ and $\overline{upd} = \mathsf{snd}$, but it is not the only comodel. It is the case however that, for any comodel, $C \cong S \times C'$ for some set C'. The C' projection of the runner's state cannot be looked up, cannot be overwritten by computations.

[7] More precisely, the bidirectional transformations term would be 'very well-behaved lens', but from our perspective the weaker (less well behaved) structures do not deserve the name 'lens'.

[8] In that context, S is called the view state set and C the source state set.

[9] This is not the minimal presentation, but the simplest one. The minimal one has the same operations, but three (more involved) equations.



where *upd* and the equations involving it can also be written in uncurried form.

The monad is $T\,X = T_0\,X/{\sim_X^*}$ where $T_0\,X$ and $\sim_X$ are defined inductively by

$$\frac{x : X}{\mathsf{var}\,x : T_0\,X} \quad \frac{f : T_0\,X \Leftarrow S}{\mathsf{lkp}\,f : T_0\,X} \quad \frac{p : P \quad c : T_0\,X}{\mathsf{upd}\,(p, c) : T_0\,X}$$

(so that $T_0\,X \cong \mu Z.\, X + (Z \Leftarrow S) + P \times Z$) and

$$\frac{\forall s.\, f\,s \sim_X f'\,s}{\mathsf{lkp}\,f \sim_X \mathsf{lkp}\,f'} \quad \frac{c \sim_X c'}{\mathsf{upd}\,(p, c) \sim_X \mathsf{upd}\,(p, c')} \quad \frac{}{c \sim_X \mathsf{lkp}\,(\lambda s.\, c)} \quad \frac{}{\mathsf{lkp}\,(\lambda s'.\, \mathsf{lkp}\,(\lambda s.\, f\,s\,s')) \sim_X \mathsf{lkp}\,(\lambda s.\, f\,s\,s)}$$

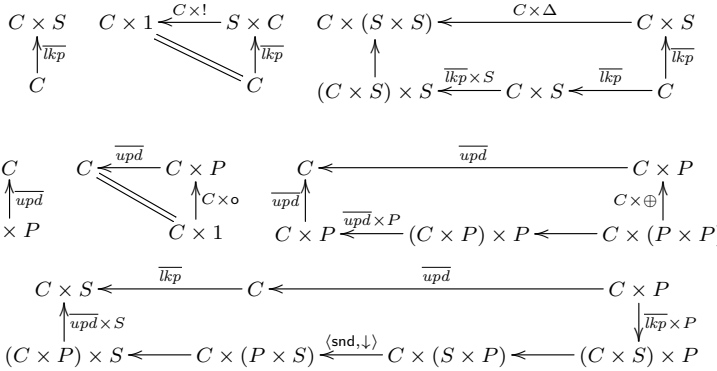
$$\frac{}{c \sim_X \mathsf{upd}\,(\mathsf{o}, c)} \quad \frac{}{\mathsf{upd}\,(p, \mathsf{upd}\,(p', c)) \sim_X \mathsf{upd}\,(p \oplus p', c)} \quad \frac{}{\mathsf{upd}\,(p, \mathsf{lkp}\,f) \sim_X \mathsf{lkp}\,(\lambda s.\, \mathsf{upd}\,(p, f\,(s \downarrow p)))}$$

Since the equations allow us to present any element of $T\,X$ uniquely in the normal form $\mathsf{lkp}\,(\lambda s.\, \mathsf{upd}\,(g\,s, \mathsf{var}\,(h\,s)))$ for some $\langle g, h\rangle : (P \times X) \Leftarrow S$, we get that $T\,X \cong (P \times X) \Leftarrow S$ whereby

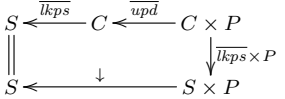
$$\eta_X\,x = \lambda s.\, (\mathsf{o}, x)$$
$$\mu_X\,f = \lambda s.\, \mathsf{let}\ (p, g) \leftarrow f\,s; (p', x) \leftarrow g\,(s \downarrow p)\ \mathsf{in}\ (p \oplus p', x)$$

We have called this monad the update monad for S, $(P, \mathsf{o}, \oplus), \downarrow$ [2]. Update monads are exactly the compatible compositions of reader and writer monads—distributive laws between them are in bijections with right actions.

A comodel is a set C with functions $\overline{lkp}$ and $\overline{upd}$ such that



Splitting $\overline{lkp} = \langle \overline{lkpn}, \overline{lkps}\rangle$, we see that the 1st equation just says $\overline{lkpn} = \mathsf{id}_C$, making $\overline{lkpn}$ redundant. This makes the 2nd equation tautological and simplifies the 5th to



We have previously christened these structures update lenses [1], they are a refinement of state-based lenses. An update lens is a machine that responds to lookups (without changing its state) and listens to updates.

The corresponding comonad is $D\,X = D_0\,X \mid ok_X$ where $D_0\,X$ and ok_X are defined coinductively by

$$\frac{c : D_0\,X}{\overline{\mathsf{var}}\,c : X} \quad \frac{c : D_0\,X}{\overline{\mathsf{lkps}}\,c : S} \quad \frac{c : D_0\,X \quad p : P}{\overline{\mathsf{upd}}\,(c,p) : D_0\,X}$$

(so that $D_0\,X = \nu Z.\, X \times S \times (P \Rightarrow Z)$) and

$$\frac{ok_X\,c}{ok_X\,(\overline{\mathsf{upd}}\,(c,p))} \quad \frac{ok_X\,c}{c = \overline{\mathsf{upd}}\,(c,\mathsf{o})} \quad \frac{ok_X\,c}{\overline{\mathsf{upd}}\,(\overline{\mathsf{upd}}\,(c,p),p') = \overline{\mathsf{upd}}\,(c,p \oplus p')} \quad \frac{ok_X\,c}{\overline{\mathsf{lkps}}\,(\overline{\mathsf{upd}}\,(c,p)) = \overline{\mathsf{lkps}}\,c \downarrow p}$$

All information available about an element $[\,]$ of $D\,X$ is captured in the universal observation $(\overline{\mathsf{lkps}}\,[\,], \lambda p.\, \overline{\mathsf{var}}\,(\overline{\mathsf{upd}}\,([\,],p))) : S \times (P \Rightarrow X)$, which tells us that $D\,X \cong S \times (P \Rightarrow X)$ (the coupdate comonad) whereby

$$\begin{aligned} \varepsilon_X\,(s,v) &= v\,\mathsf{o} \\ \delta_X\,(s,v) &= (s, \lambda p.\,(s \downarrow p, v\,(p \oplus p'))) \end{aligned}$$

4.4 Continuations

Continuation monads have no rank, so our analysis in terms of Lawvere theories does not apply. However it it is easy to check directly that they cannot have non-trivial runners (i.e., runners with a non-empty carrier).

Indeed, fix a nonempty set R. A runner structure on a set C would be a monad morphism between the continuation monad for R and the state monad for C, so a family of maps $\theta : \forall X.\,(R \Leftarrow X) \Rightarrow R \to (C \times X) \Leftarrow C$. Consider θ_0. The set $(0 \Rightarrow R) \Rightarrow R$ is obviously inhabited (for any element r of R, it contains $\lambda f.\, r$), but for the set $(C \times 0) \Leftarrow C$ to be inhabited we need a function $C \to 0$, which can only exist if $C = 0$.

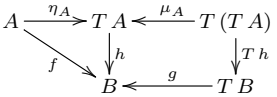
5 Running vs. handling

Runners bear some similarity to Plotkin and Pretnar's *handlers* [12], but they are not the same. Let us spell out the exact relationship.

Broadly speaking, both are about specifying ways to extract a value from a computation.

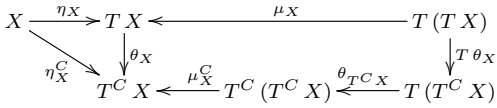
Handling is based on the fact that, for any set A, $(T\,A, \mu_A : T\,(T\,A) \to T\,A)$ is the free (T,η,μ)-algebra on A, with $\eta_A : A \to T\,A$ as the associated injection.

Spelled out, this means that, given two sets A, B, a map $f : A \to B$ and a (T,η,μ)-algebra structure $g : T\,B \to B$, we have a unique map $h : T\,A \to B$ making the diagrams



commute.

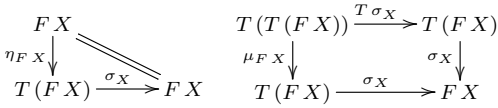
Running, at the same time, is based on the observation that any coalgebra $(C, g : C \to D\,C)$ of a suitable comonad (D, ε, δ) induces a unique monad morphism between the given monad (T, η, μ) and the state monad (T^C, η^C, μ^C). This is to say that we have a unique natural transformation θ satisfying



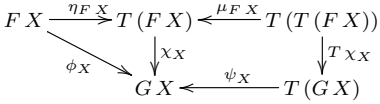
The important differences are these. First, handlers are monomorphic and any codomain is possible: a handler defines map from $T\,A$ to B for some fixed sets A, B. Runners are polymorphic, but the codomain is restricted to a specific form: a runner gives a family of functions from $T\,X$ to $T^C\,X$ for a fixed set C. Second, the data inducing handlers and runners are different.

Runners are an instance of handlers, but not in a very useful way: as a circular unique existence property rather than a direct one. Indeed, in the diagram above we have rendered the pentagon stating the condition that θ_X sends μ_X to μ^C_X in a layout suggesting that the composite map $\xi_X = \mu^C_X \circ \theta_{T^C X}$ might be a (T, η, μ)-algebra structure on $T^C\,X$, and it is easily verified to be so. But ξ_X is defined in terms of $\theta_{T^C X}$, i.e., another component of θ. So we cannot say that the algebra morphism θ_X is induced by an independently given algebra structure ξ_X.

Modulo this reservation, due to their polymorphic nature, runners are actually more than just handlers, they are *uniform handlers*. A general definition of a uniform handler proceeds from natural transformations as monad algebra structures. Say that a set functor F with a natural transformation $\sigma : T \cdot F \to F$ is a (T, η, μ)-algebra (or a *left module*), if it also meets the conditions



It is now easy to check that the free (T, η, μ)-algebra on F is $(T \cdot F, \mu \cdot F : T \cdot T \cdot F \to T \cdot F)$, with $\eta \cdot F : F \to T \cdot F$ as the associating injection. Accordingly, for any functor G, a natural transformation $\phi : F \to G$ and a (T, η, μ)-algebra structure $\psi : T \cdot G \to G$, we have a unique natural transformation $\chi : T \cdot F \to G$ such that



A runner θ is now a uniform handler for $F = \mathsf{Id}$, $G = T^C$, $\phi = \eta^C$, $\psi = \mu^C \circ (\theta \cdot T^C)$.

All of these considerations apply of course to any monad morphism from (T, η, μ), e.g., the morphism to the continuation monad on the the carrier of a (T, η, μ)-algebra discussed in Section 3—a monad morphism is always a monad algebra morphism.

6 Conclusion and future work

We showed when a computation in a given monad can be mapped into a stateful computation: it is when the state set carries a comodel of the corresponding Lawvere theory (a coalgebra of a suitable comonad). We find this to be a nice small new application of comodels (cf. the discussion by Behrisch et al. [3] on whether comodels are the "correct" dual of models). We also believe that it gives some new insight into the mechanics of different monadic/algebraic notions of effects, especially in regard to the impact the degree of abstractness (i.e., how much detail of effects we want to observe). For nondeterminism, for example, we saw that some approaches that make perfect sense for denotational semantics are not operational at all and there are simple mathematical reasons why this has to be so. We find it curious that state monads turn out to have a special role in connecting models and and comodels. Something similar appears in Møgelberg and Staton's work [7] on every monad being a linear state monad (under a certain viewpoint).

In future work, we intend to study sufficient conditions for a monad morphism to a state monad to be mono (so the stateful computation can capture all information in a given computation). We also plan to consider other target monads, in particular, combinations of state monads with other monads such as exception monads, and to relate this work and Plotkin and Power's [11] tensor of a comodel and a model.

Acknowledgment

I am most grateful to Gordon Plotkin, Alex Simpson, Tom Schrijvers and Nicolas Wu for the comments they made in response to my talks on this material as well as to the MFPS reviewers for their feedback. This research was supported by the ERDF funded projects EXCS and Coinduction, the Estonian Ministry of Education and Research institutional research grant no. IUT33-13 and the Estonian Science Foundation grant no. 9475.

References

[1] Ahman, D. and T. Uustalu, *Coalgebraic update lenses,* in: B. Jacobs, A. Silva, and S. Staton, eds., "Proc. of 30th Conf. on Mathematical Foundations of Programming Semantics, MFPS XXX (Ithaca, NY, June 2014)," Electron. Notes in Theor. Comput. Sci. **308**, Elsevier, 2014, pp. 25–48.

[2] Ahman, D. and T. Uustalu, *Update monads: cointerpreting directed containers,* in: R. Matthes and A. Schubert, eds., "Proc. of 19th Conf.d on Types for Proofs and Programs, TYPES 2013 (Toulouse, Apr. 2013)," Leibniz Int. Proc. in Inform. **26**, Dagstuhl Publishing, 2014, pp. 1–23.

[3] Behrisch, M., S. Kerkhoff, and J. Power, *Category theoretic understandings of universal algebra and its dual: monads and Lawvere theories, comonads and what?,* in: U. Berger and M. Mislove, eds., "Proc. of 28th Conf. on Mathematical Foundations of Programming Semantics, MFPS XXVIII (Bath, June 2012)," Electron. Notes in Theor. Comput. Sci. **286**, Elsevier, 2012, pp. 5–16.

[4] Foster, J. N., M. B. Greenwald, J. T. Moore, B. C. Pierce, and A. Schmitt, *Combinators for bidirectional tree transformations: a linguistic approach to the view-update problem,* ACM Trans. on Program. Lang. and Syst. **29**(3), 2007, article 17.

[5] Hyland, M., P. B. Levy, G. Plotkin, and J. Power, *Combining algebraic effects with continuations,* Theor. Comput. Sci. **375**(1–3), 2007, pp. 20–40.

[6] Hyland, M. and J. Power, *The category theoretic understanding of universal algebra: Lawvere theories and monads,* in: L. Cardelli, M. Fiore, and G. Winskel, eds., "Computation, Meaning, and Logic: Articles Dedicated to Gordon Plotkin," Electron. Notes in Theor. Comput. Sci. **172**, Elsevier, 2007, pp. 437–458.

[7] Møgelberg, R. E. and S. Staton, *Linear usage of state,* Log. Methods in Comput. Sci. **10**(1), 2014, article 17.

[8] Moggi, E., *Notions of computation and monads*, Inf. and Comput. **93**(1), 1991, pp. 55–92.

[9] O'Connor, R., *Functor is to lens as applicative is to biplate: introducing multiplate,* arXiv preprint 1103.2841, 2011. (Paper presented at WGP 2011, Tokyo, Sept. 2011.)

[10] Plotkin, G. and J. Power, *Notions of computation determine monads,* in: M. Nielsen and U. Engberg, "Proc. of 5th Int. Conf. on Foundations of Software Science and Computation Structures, FoSSaCS 2002 (Grenoble, Apr. 2002)," Lect. Notes in Comput. Sci. **2303**, Springer, 2002, pp. 342–356. Springer (2002)

[11] Plotkin, G. and J. Power, *Tensors of comodels and models for operational semantics,* in: A. Bauer and M. Mislove, eds., "Proc. of 30th Conf. on Mathematical Foundations of Programming Semantics, MFPS XXIV (Philadelphia, PA, May 2008)," Electron. Notes in Theor. Comput. Sci. **218**, Elsevier, 2008, pp. 295–311.

[12] Plotkin, G. and M. Pretnar, *Handling algebraic effects,* Log. Methods in Comput. Sci. **9**(4), 2013, article 23.

[13] Power, J. and O. Shkaravska, *From models to comodels: state and arrays,* in: J. Adámek and S. Milius, eds., "Proc. of 7th Int. Wksh. on Coalgebraic Methods in Computer Science, CMCS 2004 (Barcelona, March 2004)," Electron. Notes in Theor. Comput. Sci. **106**, Elsevier, 2004, pp. 297–314.