Elementary Dependent Sweedler Runners

Part A: A Gallery of Small Machines

Abstract

In the ordinary Sweedler tutorial, the measuring comonads $\{\mathbb{N}y, \mathrm{Id}\}$, $\{\mathbb{N}y, \mathbb{N}y\}$, and the Moore and reader examples turned abstract runners into familiar machines. This part repeats that strategy when operations and continuations have *worlds* or *modes*. We calculate alternating systems, time-dilating implementations, reversible systems, dependent sensors and Moore machines, a read/EOF protocol, path-constrained file implementations, nondeterministic automata and finite-lottery machines. This part is self-contained given the ordinary Sweedler tutorial: all dependent notation is explained through the machines. A companion Part B develops the categorical structure behind these computations.

1 The two-world laboratory

A dependent machine has more than one sort of state. For a finite set W of worlds, write $\mathbf{Set}^W$ for the category of families $X = (X_w)_{w \in W}$. A source program at w must resume a continuation at a prescribed world; the residual state after that command must lie in the same world as the continuation. This is the extra discipline that we want to see in examples.

We shall write $T \rightsquigarrow^S R$ for a runner interpreting source theory T in target theory R, with residual state family S. A runner executes each source command using a target program, chooses the response continuation, and returns a residual state of the required world. The usual notation $T \rightsquigarrow^S R$ and $\{T, R\}$ is retained from the ordinary Sweedler tutorial [2]. A coalgebra for the measuring comonad is a runner; rather than construct this comonad abstractly, we calculate the machine data directly from commands and their laws. When useful, we describe its cofree behaviour as the complete future unfolding.

Start with $W = \{0, 1\}$. Three elementary monads will recur:

$$\begin{aligned} I(X)_w &= X_w, \\ K(X)_w &= X_0 + X_1, \\ L(X)_w &= \coprod_{v \in W} \mathrm{Path}(w, v) \times X_v. \end{aligned}$$

Here I is the identity monad; K is the path monad of the indiscrete category on $\{0, 1\}$, so it has one arrow for every ordered pair; and L is the free path monad of

$$0 \xrightarrow{a} 1 \xrightarrow{b} 0.$$

The target I cannot change worlds. The target K can change worlds but retains no information about how. The target L remembers the entire alternating path, equivalently its length and endpoints. Thus $\mathrm{Path}(0, 1)$ and $\mathrm{Path}(1, 0)$ are indexed by positive odd integers, while same-world paths have even lengths, including zero.

2 Three versions of unary dynamics

2.1 $\{L, I\}$: an impossible change of world

A runner must interpret $a : 0 \to 1$ by an I-computation starting at 0 and resuming at 1. Such a computation does not exist. Thus S_0 must be empty. The command b similarly forces $S_1 = \varnothing$. There is only the empty runner, and the measuring comonad is the constant empty-family comonad:

$$\{L, I\}(X) = (\varnothing, \varnothing).$$

Contrast this with $\{\mathbb{N}y, \mathrm{Id}\}$ on Set, whose coalgebras are arbitrary endomorphisms $S \to S$. An ordinary unary command may be interpreted by doing nothing to the target computation and changing only residual state; a *typed world-changing* unary command cannot.

2.2 $\{L, K\}$: alternating dynamical systems

Now K has exactly one arrow $0 \to 1$ and one $1 \to 0$. The only choices are the new residual states:

$$d_a : S_0 \to S_1, \qquad d_b : S_1 \to S_0.$$

The coalgebras are familiar two-mode deterministic dynamics. The one-step functor is

$$H(S)_0 = S_1, \qquad H(S)_1 = S_0.$$

The cofree measuring comonad records alternating state labels:

$$\{L, K\}(X)_0 = X_0 \times X_1 \times X_0 \times \cdots,$$
$$\{L, K\}(X)_1 = X_1 \times X_0 \times X_1 \times \cdots.$$

Its counit reads the first label and its comultiplication lists all successive tails. This is the typed analogue of the stream comonad $X^{\mathbb{N}}$.

2.3 $\{L, L\}$: state-dependent time dilation

A translation of $a : 0 \to 1$ into the target L is an odd-length alternating path, together with a new residual state. Thus a runner is given by four functions:

$$k_a : S_0 \to \mathbb{N}_{\mathrm{odd}}, \quad d_a : S_0 \to S_1,$$
$$k_b : S_1 \to \mathbb{N}_{\mathrm{odd}}, \quad d_b : S_1 \to S_0.$$

Here $\mathbb{N}_{\mathrm{odd}} = \{1, 3, 5, \ldots\}$. For a source cycle ab, the total target length and final state are

$$K_{ab}(s) = k_a(s) + k_b(d_a(s)), \qquad D_{ab}(s) = d_b(d_a(s)).$$

The one-step coalgebra functor has components $H(S)_0 = \mathbb{N}_{\mathrm{odd}} \times S_1$ and $H(S)_1 = \mathbb{N}_{\mathrm{odd}} \times S_0$; its cofree comonad consists of alternating streams of labels and positive odd step counts. Unlike the ordinary $\{\mathbb{N}y, \mathbb{N}y\}$, one source step here cannot be implemented by zero target steps: it must change worlds.

3 Equations: reversible systems and dependent sensors

3.1 $\{K, K\}$: a specified bijection

In K, the unique arrow $0 \to 1$ is inverse to the unique arrow $1 \to 0$. The runner laws therefore require

$$d_{10}d_{01} = \mathrm{id}_{S_0}, \qquad d_{01}d_{10} = \mathrm{id}_{S_1}.$$

A runner is a pair of sets with a chosen bijection. Its measuring comonad is

$$\{K, K\}(X)_0 = \{K, K\}(X)_1 = X_0 \times X_1.$$

The comonad's coalgebras recover the bijections. This contrasts with $\{L, K\}$, where the two transitions may be arbitrary functions. The source equations, not the target path equations alone, force reversibility. Indeed $\{K, I\}$ has only the empty runner: the target cannot implement a world change. The same holds for $\{K, L\}$: the source equation saying that the $0 \to 1 \to 0$ composite is the identity would require the sum of two strictly positive odd target lengths to be zero. Thus a target that records nonzero path length cannot implement these inverse source commands on nonempty residual fibres.

3.2 A one-sided protocol equation

Impose only the source equation $a; b = \mathrm{id}_0$, leaving $b; a$ unconstrained. A runner into K must satisfy $d_b d_a = \mathrm{id}_{S_0}$: thus S_0 is a retract of S_1, rather than requiring a bijection. With L as target, however, a nonempty S_0 is impossible, because the interpreted composite would have strictly positive even length. Then the required map $d_b : S_1 \to S_0 = \varnothing$ forces $S_1 = \varnothing$ as well: the entire runner is empty. This is another backward-propagation instance of viability, and the simplest example of a protocol law constraining the implementation beyond mere world typing.

3.3 A dependent reader: a sensor in each mode

Choose an observation set O_w for each world and take the fibrewise reader monad

$$R_O(X)_w = X_w^{O_w}.$$

Its comodels into I consist of observation maps $o_w : S_w \to O_w$; the reader equations force observation not to modify residual state. The same answer is obtained with target K: every continuation of get_w lies at w, and the one-step target computation can only select the identity arrow $w \to w$. Thus in this comparison typing, rather than the reader equations, excludes a change of world:

$$\{R_O, I\}(X)_w \cong \{R_O, K\}(X)_w \cong O_w \times X_w.$$

The reader equations do matter when the target is L, which permits nonidentity even-length excursions from w back to w. In that case the discard equation $\mathsf{get}_w(\lambda o.\, x) = x$ forces the excursion length to be zero and the residual update to be the identity. Thus $\{R_O, L\}(X)_w \cong O_w \times X_w$ as well. The counit is $(o, x) \mapsto x$ and comultiplication is $(o, x) \mapsto (o, (o, x))$. A sensor might report a status at world 0 and an integer measurement at world 1; the observation type itself depends on the mode.

3.4 Dependent Moore machines

Adjoin to these reader operations the unary commands $a : 0 \to 1$ and $b : 1 \to 0$, retaining the reader equations but imposing no equations on the commands. Call the source monad T_M. A runner $T_M \rightsquigarrow^S K$ is exactly

$$o_0 : S_0 \to O_0, \quad d_a : S_0 \to S_1, \qquad o_1 : S_1 \to O_1, \quad d_b : S_1 \to S_0.$$

In other words, it is a two-mode Moore machine: each mode has its own state set and observations, and its command leads to the other mode. The one-step coalgebra functor is

$$H_M(S)_0 = O_0 \times S_1, \qquad H_M(S)_1 = O_1 \times S_0.$$

Starting at world 0, a cofree behaviour records the alternation

$$(o_0, x_0), (o_1, x_1), (o_2, x_2), \ldots, \quad (o_{2n}, x_{2n}) \in O_0 \times X_0, \quad (o_{2n+1}, x_{2n+1}) \in O_1 \times X_1.$$

These are just Moore streams with types attached to successive positions. More generally, if world w permits commands $a \in A_w$ with destinations $t(a)$, the corresponding typed Moore machine is a coalgebra for $H(S)_w = O_w \times \prod_{a \in A_w} S_{t(a)}$. This is the full dependent analogue of a Moore machine with several inputs. The alternating example makes its types especially easy to see.

3.5 Dependent Mealy machines

Our next examples are instances of a familiar kind of machine: a Mealy machine whose available commands, responses, and successor-state types depend on its current world.

Let W be a set of worlds. For each $w \in W$, suppose we are given a set A_w of commands. A command $a \in A_w$ has a set $B_w(a)$ of possible responses, and each response $b \in B_w(a)$ determines a destination world

$$d(w, a, b) \in W.$$

A *dependent Mealy machine* consists of state sets $(S_w)_{w \in W}$ and transition maps

$$\delta_{w,a} : S_w \longrightarrow \coprod_{b \in B_w(a)} S_{d(w,a,b)}.$$

Thus the machine, when presented with a command a in state $s \in S_w$, chooses a response b and moves to a state whose type is determined by that response.

Equivalently, these machines are coalgebras for

$$F(S)_w = \prod_{a \in A_w} \coprod_{b \in B_w(a)} S_{d(w,a,b)}. \tag{DM}$$

When W is a singleton, this reduces to the ordinary Mealy-machine functor

$$F(S) = \prod_{a \in A} (B(a) \times S).$$

The Sweedler calculation. Form the free dependent polynomial theory $T = P^*$ on the signature

$$P(X)_w = \coprod_{a \in A_w} \prod_{b \in B_w(a)} X_{d(w,a,b)}.$$

The source program supplies a continuation for every possible response to a command.

As residual theory, take the monad K_W of the indiscrete category on W:

$$K_W(X)_w = \coprod_{v \in W} X_v.$$

There is exactly one implementation path between any two worlds. Consequently, a runner interpreting a must choose a response b and a residual state in the corresponding destination world, but has no further implementation-path choices.

It follows directly that

$$\boxed{\mathrm{Coalg}(\{T, K_W\}) \simeq \mathrm{Coalg}(F).}$$

The equivalence respects the underlying families of states and their morphisms. Thus dependent Mealy machines are precisely the comodels arising from this elementary pair of dependent theories.

This also explains the role of the residual theory. Replacing K_W by a path monad allows different implementation paths and costs; replacing it by a nondeterministic or probabilistic target allows several possible outcomes or lotteries over outcomes.

Example: reading until EOF. Take

$$W = \{\mathsf{Open}, \mathsf{EOF}\}.$$

At Open there is one command, read, with responses $\mathrm{Data}(d)$ for $d \in D$, and EOF. A data response returns to Open, whereas EOF changes the world to EOF. There are no commands at EOF.

The corresponding dependent Mealy transition is

$$S_{\mathsf{Open}} \longrightarrow (D \times S_{\mathsf{Open}}) + S_{\mathsf{EOF}}.$$

This is the possibly terminating stream machine studied next. Unlike an ordinary Mealy machine, its response determines the type of the successor state. In particular, another read cannot be issued after EOF.

Relation to verified Mealy machines. The dependence considered here is operational: commands, responses, and successor-state types vary with the world. Aberlé's dependent Mealy machines ([1], §6.1) add a different, complementary kind of dependence. They carry command- and response-indexed verification evidence over an underlying Mealy behaviour. Such evidence can be incorporated into a proof-relevant ambient category, as in our earlier work with the arrow topos. The present calculation requires no certification structure; it describes the underlying typed machines themselves.

4 Branching: read a datum or reach EOF

Let $W = \{O, E\}$, standing for Open and EOF, and let D be a set of data values. A program can issue one *read request* while Open. It must supply a continuation for every possible datum $d \in D$, returning to Open, and a continuation for EOF, entering world E. There are no reads at E.

The signature is therefore

$$P_{\mathsf{read}}(X)_O = X_O^D \times X_E, \qquad P_{\mathsf{read}}(X)_E = 0.$$

Put $T_{\mathsf{read}} = P_{\mathsf{read}}^*$, the free monad. The distinction between $X_O^D \times X_E$ and $(D \times X_O) + X_E$ matters: the first is *one command with many possible responses*; the second is a family of unary commands chosen by the program. In a runner, the implementation chooses which continuation is resumed.

4.1 $\{T_{\mathsf{read}}, K\}$: a possibly terminating stream producer

Use the indiscrete-category monad $K(X)_w = X_O + X_E$. A runner is determined by

$$r : S_O \longrightarrow (D \times S_O) + S_E.$$

At each Open state, the reader either produces a datum and a new Open state, or announces EOF and enters an EOF state. This is a deterministic, possibly terminating stream producer. Its one-step functor is

$$H(S)_O = (D \times S_O) + S_E, \qquad H(S)_E = 1.$$

The cofree measuring comonad $G = \{T_{\mathsf{read}}, K\}$ records both finite and infinite futures:

$$G(X)_E = X_E,$$
$$G(X)_O \cong X_O \times ((D \times G(X)_O) + G(X)_E).$$

Here the final solution is intended. Explicitly, its Open component is

$$\left(\coprod_{n \geq 0} (X_O \times D)^n \times X_O \times X_E \right) + (X_O \times D)^{\mathbb{N}}.$$

A finite branch carries labels for every Open state and for the terminal EOF state; an infinite branch never reports EOF. The counit reads the current label and comultiplication records successive future behaviours.

4.2 $\{T_{\mathsf{read}}, I\}$: EOF is unavailable

The identity target cannot return an EOF-world continuation. It may, however, select a data continuation and remain at Open. Consequently

$$r : S_O \longrightarrow D \times S_O, \qquad \{T_{\mathsf{read}}, I\}(X)_O = (X_O \times D)^{\mathbb{N}}, \quad \{T_{\mathsf{read}}, I\}(X)_E = X_E.$$

The EOF state set is unconstrained, but no Open execution can reach it. This contrasts with an *obligatory* world-changing unary command, which had no identity-target runner on a nonempty state set. A branching request may offer a world change without ever choosing it.

4.3 $\{T_{\mathsf{read}}, L\}$: implementation lengths

Let L now be the free alternating-path monad on $O \rightleftarrows E$. An outcome returning data at Open uses an even-length path; an EOF outcome uses an odd-length path. A runner is exactly

$$r : S_O \longrightarrow (\mathbb{N}_{\text{even}} \times D \times S_O) + (\mathbb{N}_{\text{odd}} \times S_E),$$

where $\mathbb{N}_{\text{even}} = \{0, 2, 4, \ldots\}$. This is a possibly terminating stream machine that records how many target steps implement each read. A data response may take zero steps, while EOF requires a positive odd number of steps. The restriction comes from world typing alone.

5 A three-world file protocol

The two-world reader sits naturally inside a more complete typestate machine. Let

$$W = \{C, O, E\} = \{\mathsf{Closed}, \mathsf{Open}, \mathsf{EOF}\}.$$

The source T_{file} is freely generated by the following commands:

Current world	Command	Possible continuation worlds
C	open	O
O	read	data $d \in D$ at O, or EOF at E
O	close	C
E	reset	C

With the indiscrete-category target K, a runner has exactly the familiar machine data

$$\begin{aligned} \mathsf{open} &: S_C \to S_O, & \mathsf{close} &: S_O \to S_C, \\ \mathsf{read} &: S_O \to (D \times S_O) + S_E, & \mathsf{reset} &: S_E \to S_C. \end{aligned}$$

The corresponding coalgebra functor is

$$\begin{aligned} H(S)_C &= S_O, \\ H(S)_O &= S_C \times \big((D \times S_O) + S_E\big), \\ H(S)_E &= S_C. \end{aligned}$$

Why a product at Open? Because the machine must interpret *both* available commands, read and close, at each Open state. Why a coproduct inside read? Because a given execution of read selects *one* response.

5.1 Restrict the target paths

We can learn a great deal without changing the source. For a category $\mathcal{C}$ on the three worlds, put

$$R_{\mathcal{C}}(X)_w = \coprod_{v \in W} \mathcal{C}(w, v) \times X_v.$$

A runner into $R_{\mathcal{C}}$ consists of

$$\begin{aligned} \mathsf{open} &: S_C \to \mathcal{C}(C, O) \times S_O, \\ \mathsf{close} &: S_O \to \mathcal{C}(O, C) \times S_C, \\ \mathsf{reset} &: S_E \to \mathcal{C}(E, C) \times S_C, \\ \mathsf{read} &: S_O \to \big(\mathcal{C}(O, O) \times D \times S_O\big) + \big(\mathcal{C}(O, E) \times S_E\big). \end{aligned}$$

For the forward-only target graph $C \to O \to E$, no path returns from O to C or from E to C. The commands close and reset force $S_O = S_E = \varnothing$, and then open forces $S_C = \varnothing$. Only the empty runner survives.

More interestingly, take the freely generated graph

$$C \rightleftarrows O \longrightarrow E,$$

where E has no outgoing nonidentity paths. The target does have arrows $O \to E$ (indeed infinitely many), but no arrows $E \to C$. Therefore reset forces $S_E = \varnothing$, killing the EOF-response summand of read *even though the target can reach EOF*. Open, close and data-producing read remain possible. Paths $C \to O$, $O \to C$ and $O \to O$ are counted by $\mathbb{N}$ (the number of complete two-step cycles). The nonempty runner data reduce to

$$\begin{aligned} \mathsf{open} &: S_C \to \mathbb{N} \times S_O, \\ \mathsf{close} &: S_O \to \mathbb{N} \times S_C, \\ \mathsf{read} &: S_O \to \mathbb{N} \times D \times S_O. \end{aligned}$$

The implementation may repeatedly read data but cannot report EOF. For its cofree comonad G, one obtains

$$\begin{aligned} G(X)_C &\cong X_C \times \mathbb{N} \times G(X)_O, \\ G(X)_O &\cong X_O \times (\mathbb{N} \times G(X)_C) \times (\mathbb{N} \times D \times G(X)_O), \\ G(X)_E &= \varnothing. \end{aligned}$$

This is a tree of future interactions, with a branch for each available command, but no EOF states. If $D = \varnothing$, the Open fibre must also be empty, and so must the Closed fibre.

The absence of EOF is enforced indirectly: an EOF state cannot discharge its subsequent reset obligation, so no such residual state can occur. This is an instance of the viability calculation below. It is a simple, type-level safety condition; it does not automatically prove the safety of internal low-level states traversed along a target path.

6 Which worlds can support a runner? A viability test

The preceding examples have a common explanation. For a free dependent polynomial source $T = P^*$ and a deterministic path target $R_{\mathcal{C}}$, write A_w for the commands at world w, E_a for the response positions of command a, and $s(e)$ for a response's destination world. Define an operator on subsets $V \subseteq W$ by

$$\Phi(V) = \{w \in W : \ \forall a \in A_w \ \exists e \in E_a \text{ such that } s(e) \in V \text{ and } \mathcal{C}(w, s(e)) \neq \varnothing\}.$$

Proposition 1 (Viable worlds for free source theories)**.** *Let $T = P^*$ be the* free *dependent polynomial monad, and let $R_{\mathcal{C}}$ be the path monad of a category with object set W. The worlds which can carry a nonempty residual-state fibre in some runner $T \rightsquigarrow R_{\mathcal{C}}$ form $\nu\Phi$. Equivalently, the support of the final unlabelled one-step coalgebra is $\nu\Phi$. For the cofree coalgebra with labels X, its support is exactly*

$$\nu\big(V \mapsto \Phi(V) \cap \operatorname{supp}(X)\big), \qquad \operatorname{supp}(X) = \{w \mid X_w \neq \varnothing\}.$$

Proof. If $S_w \neq \varnothing$, every command $a \in A_w$ must select a response position e, a target path $w \to s(e)$, and a state in $S_{s(e)}$. Thus the support $\operatorname{supp}(S)$ satisfies $\operatorname{supp}(S) \subseteq \Phi(\operatorname{supp}(S))$. Conversely, given a post-fixed set $V \subseteq \Phi(V)$, set $S_w = 1$ for $w \in V$ and $S_w = \varnothing$ otherwise. Choosing for each command an allowed position and target path gives a runner supported on V. Knaster–Tarski identifies $\nu\Phi$ as the union of all post-fixed subsets; the stated supports follow (with choice for the selections). For the labelled case, require in addition that each supported world carry an X-label and apply the same argument to $V \mapsto \Phi(V) \cap \operatorname{supp}(X)$. □

The quantifiers are important: *every* source command must have *some* implementable response. Thus an obligatory world change may kill a world, whereas an EOF response merely offered

by read need never be selected. In the file example, non-viability propagates backwards from E because an EOF state cannot implement reset. This is a discrete viability kernel, analogous to the viability constructions of control theory [3]. The proposition is specific to *free source theories*. Source equations can shrink the actual viable support below $\nu\Phi$: for example $\{K, L\}$ in Section 3 has empty residual fibres despite both worlds passing the one-step test. The equation imposing a zero-length composite cannot be detected by Φ.

A nullary source command ($E_a = \varnothing$) makes its world non-viable for a total path target: no response position can be selected. For the nonempty finite-powerset target, viability has the same $\forall\exists$ condition: a nonempty set of outcomes exists exactly when at least one outcome is available. For the finite-powerset monad allowing $\varnothing$, every command may choose the empty result; consequently every world becomes viable, although some operations may deadlock. This is the precise distinction between total and failure-permitting implementations.

7 A context restriction: read and reset as one program

Consider again the file protocol with the three modes Closed, Open and EOF. Suppose we want programs whose *visible* initial and final modes are Closed or Open; EOF is no longer an externally visible mode. There are two possible constructions.

Restrict primitive commands first. Retain a primitive operation only when *every* response continuation has a mode in $\{C, O\}$. The operation `read` must then be discarded: one possible response, EOF, asks for a continuation outside those modes. Forming programs from the remaining commands can never restore `read`.

Construct complete programs first. Instead retain any well-typed program of the original three-mode protocol whose initial mode and all *variable-return* modes belong to $\{C, O\}$. Such a program may visit EOF internally, provided it handles that response before returning. For instance,

$$\mathsf{read}\big((d \mapsto \text{return in Open}),\ \mathsf{reset}; \text{return in Closed}\big)$$

is admissible. If a datum arrives, it returns in Open; if EOF arrives, it resets and returns in Closed. Both variable continuations lie in the visible modes.

Every program assembled from the restricted primitive commands is included among the complete programs with visible returns, but not conversely: the displayed read-and-reset program is missing from the first construction. This is not a subtlety about mathematical notation. It says that *a command may be used safely in a smaller external interface if the program handles its temporary excursion through a hidden mode.* The next part explains this difference categorically as a comparison between restricting a signature and restricting its completed programs.

8 Nondeterministic outcomes and deadlock

One construction behind several targets. Our target K lets a computation return a value tagged by any destination world, and is otherwise deterministic. The targets N and Q below do the same with nondeterministic choices and probability tickets. More precisely, they take the disjoint union of all world-indexed return types, apply the familiar ordinary effect (identity, finite nonempty powerset, or finite lottery), and retain the tags on the results. This elementary recipe explains why we can recognise many of our examples as standard machines whose outcomes obey additional world-typing conditions.

Return briefly to $W = \{O, E\}$ and T_{read}. Let $\mathcal{P}^+_{\mathrm{fin}}$ be the finite nonempty powerset monad and $q : W \to 1$ the terminal map. Define

$$N(X)_w = \mathcal{P}^+_{\mathrm{fin}}(X_O + X_E).$$

The monad unit uses singletons and multiplication takes unions, with worlds retained as coproduct tags. A runner is now a finitely branching read machine:

$$r : S_O \longrightarrow \mathcal{P}^+_{\mathrm{fin}}\big((D \times S_O) + S_E\big).$$

Unlike the deterministic example, a single state can admit both a data response and an EOF response. For instance, with $D = \{0, 1\}$,

$$r(s) = \{(\mathsf{Data}(0), t), (\mathsf{EOF}, e)\}, \qquad r(t) = \{(\mathsf{Data}(1), t)\}.$$

The one-step functor is

$$H(S)_O = \mathcal{P}^+_{\mathrm{fin}}\big((D \times S_O) + S_E\big), \qquad H(S)_E = 1.$$

The measuring comonad is its cofree comonad. Its Open component is the final solution of

$$G(X)_O \cong X_O \times \mathcal{P}^+_{\mathrm{fin}}\big((D \times G(X)_O) + X_E\big), \qquad G(X)_E = X_E.$$

It records finitely branching, possibly infinite behaviour trees, up to the equality appropriate to finite sets of successor behaviours. The singleton embedding $K \to N$ sends deterministic machines to the nondeterministic ones having a singleton set of outcomes.

There are two standard questions about an EOF hazard. *May-safety* asks whether some run can avoid EOF, while *must-safety* asks whether every run avoids it. For the displayed machine, s may avoid EOF but is not must-safe; t is must-safe. The largest must-safe subset $P \subseteq S_O$ is the greatest fixed point of

$$\Psi(P) = \{s \in S_O \mid r(s) \subseteq D \times P\}.$$

Here $P = \{t\}$. This is the usual coalgebraic safety invariant, now respecting dependent response worlds. Both this safety predicate and the viability kernel of the preceding section are greatest-fixed-point (ν) constructions.

If we replace $\mathcal{P}^+_{\mathrm{fin}}$ by the finite powerset $\mathcal{P}_{\mathrm{fin}}$, the empty outcome set is permitted. A command may then have no possible successful return: a deadlock or failed computation. In particular, a transition impossible with a total deterministic target can become implementable by the empty computation. Whether that counts as a correct implementation depends on whether the intended semantics admits failure.

For the complete file protocol, finite nonempty nondeterminism gives

$$\begin{aligned}
\mathsf{open} &: S_C \to \mathcal{P}^+_{\mathrm{fin}}(S_O),\\
\mathsf{close} &: S_O \to \mathcal{P}^+_{\mathrm{fin}}(S_C),\\
\mathsf{reset} &: S_E \to \mathcal{P}^+_{\mathrm{fin}}(S_C),\\
\mathsf{read} &: S_O \to \mathcal{P}^+_{\mathrm{fin}}((D \times S_O) + S_E).
\end{aligned}$$

Thus we have obtained an ordinary nondeterministic typestate automaton merely by changing the residual theory.

9 Finite lotteries: a probabilistic stopping point

There is a parallel probabilistic example which can be calculated without choosing a sophisticated probabilistic ambient category. Let rand be the finite *ticket* lottery monad on Set: a lottery consists of finitely many distinguishable tickets i, each with probability $p_i \geq 0$, $\sum_i p_i = 1$, and a return value x_i. Substitution flattens the tree of tickets and multiplies the probabilities along each branch. Keeping tickets distinct avoids silently identifying a lottery with a probability distribution on its set of values. The composition of probability vectors is the operad-of-simplices composition discussed by Leinster [4] and related to the entropy characterisation of Baez–Fritz–Leinster [5]. The quotient which merges tickets with equal outcomes is a monad morphism $\mathrm{rand} \to \mathcal{D}_{\mathrm{fin}}$ to finite distributions; it also pushes runners forward to distribution-valued implementations.

Use the two-world target

$$Q(X)_w = \mathrm{rand}(X_O + X_E),$$

formed by applying the ordinary finite-lottery monad to the disjoint union of the two return types. A runner from T_{read} is a map

$$\boxed{r : S_O \longrightarrow \mathrm{rand}((D \times S_O) + S_E).}$$

This is a finite-branching probabilistic stream producer with possible termination. A state may return data with some probabilities and EOF with the remaining probability.

For a concrete illustration, take Open states s, t and a single EOF state e. Let

$$r(s) = \big(\tfrac{1}{2} : (\mathsf{Data}(0), t),\ \tfrac{1}{2} : (\mathsf{EOF}, e)\big), \qquad r(t) = \big(1 : (\mathsf{Data}(1), t)\big).$$

Starting from s, EOF occurs at the first read with probability $1/2$, and otherwise the system moves to t and outputs ones indefinitely. It follows that the eventual EOF probability is $1/2$.

In general, the probability $h(s)$ of ever returning EOF is the *least* fixed point of

$$h(s) = \sum_i p_i \begin{cases} 1, & \text{ticket } i \text{ returns EOF,} \\ h(s_i), & \text{ticket } i \text{ returns data and state } s_i. \end{cases}$$

This eventual-reachability calculation is a least-fixed-point (μ) construction, in contrast with viability and universal safety, which use greatest fixed points. The least fixed point is important: a perpetual data-producing loop must have eventual EOF probability zero, not an arbitrary solution of $h = h$. The cofree measuring comonad, when taken in this finite-ticket setting, describes the corresponding probability-labelled unfolding of the machine. Here we only need its elementary runner description and the stopping-probability calculation. Questions about probability distributions over infinite behaviours, measurable spaces and stochastic verification belong to the separate probabilistic tutorial.

10 What the examples establish

The series can be summarised in a small dictionary. In each row the source specifies legal requests and typed continuations, while the target specifies what kinds of implementation computations may return.

Source T	Target R	Familiar coalgebras of $\{T, R\}$
Alternating paths L	I	Only the empty system
Alternating paths L	K	Two-mode deterministic dynamics
Alternating paths L	L	Dynamics with state-dependent step counts
Indiscrete-category K	K	A specified bijection of state fibres
Indiscrete-category K	I or L	Only the empty system
One-sided inverse law	K	A retraction of state fibres
Dependent reader R_O	I or L	Mode-indexed sensors; reader laws eliminate L-excursions
Dependent Moore T_M	K	Typed Moore machines, with $H(S)_w = O_w \times \prod_{a \in A_w} S_{t(a)}$
Read/EOF T_{read}	K	Possibly terminating deterministic streams
Read/EOF T_{read}	I	Data streams that cannot reach EOF
Read/EOF T_{read}	L	Streams with typed path lengths
Read/EOF T_{read}	N	Finitely branching nondeterministic readers
Read/EOF T_{read}	Q	Finite-lottery readers with EOF probabilities
File protocol T_{file}	$R_{\mathcal{C}}$	Path-constrained typestate machines

The key distinction is now visible in very familiar systems:

source: which continuations are offered	target: which outcomes can actually occur.

Worlds matter operationally. A same-world target cannot implement an obligatory world change; source equations can force reversibility (and make some targets impossible); a missing target path can exclude an entire residual-state fibre; and a read request can offer EOF without requiring its implementation ever to choose EOF. Nondeterminism, failure and probability then arise by changing only the target theory.

We have deliberately not used the general theory of change-of-world squares, matrix enrichment or dependent double categories. Those structures may ultimately explain how such machines compose and how certificates move between implementations. The purpose of this part has been more elementary: to show that dependent Sweedler comodels are recognisable machines long before the abstract theory is invoked. One important example remains for a future part: a file protocol implemented over a genuinely different low-level world type (for instance a block-device or buffered socket protocol). That requires a span of world sets rather than the shared-base runners used here.

References and context

References

[1] C. A. Aberlé, *Compositional Program Verification with Polynomial Functors in Dependent Type Theory* , arXiv:2604.01303

[2] D. McDermott, E. Rivas and T. Uustalu, *Sweedler Theory of Monads*, FoSSaCS 2022, LNCS 13242, pp. 428–448, doi:10.1007/978-3-030-99253-8_22.

[3] J.-P. Aubin, *Viability Theory*, Birkhäuser, 1991.

[4] T. Leinster, *Entropy and Diversity: The Axiomatic Approach*, Cambridge University Press, 2021, Chapter 12 (operads and entropy).

[5] J. C. Baez, T. Fritz and T. Leinster, *A characterization of entropy in terms of information loss*, *Entropy* 13 (2011), 1945–1957.

[6] N. Gambino and J. Kock, *Polynomial functors and polynomial monads*, *Math. Proc. Cambridge Philos. Soc.* 154 (2013), 153–192.