

Getting to Grips with Sweedler Runners for Systems

A conversational tutorial, Part II: residual effects and machine-on-machine realisation

September 2026

Aim. Part I ended with ordinary Moore machines as runners into the identity monad. Here we let a genuine residual theory remain. We begin with probability, compare it with finite nondeterministic branching, and then let the residual theory itself be another machine theory. The recurring question is: *how can the operations and equations of an abstract theory be realised using the capabilities of another theory?*

Contents

1	Where Part I left us	2
1.1	Keep the handler direction straight	3
1.2	Residual theory as machine language	4
2	Our first residual effect: $T = \mathbb{N}y$, $R = \text{rand}$	5
2.1	Several abstract ticks	6
2.2	A residual handler turns the lottery into a value	6
2.3	Expectation: a Markov operator appears	7
2.4	Entropy plus continuation cost	7
2.5	The left adjoint in this example	7
3	Free branching through probability: $F(y^O) \rightsquigarrow \text{rand}$	8
3.1	Running a whole tree	8
3.2	The induced O -ary handler operation	8
3.3	The lottery presentation can matter	9
4	Quotient to reader: the equations kill the randomness	9
4.1	The first reader equation is already decisive	9
4.2	The phenomenon is not only about ticket sensitivity	10
4.3	Handler induction becomes independent of the residual algebra	10
5	Moore machines through probability	10
5.1	Running a whole Moore computation	11
5.2	The output process can still be random	11
5.3	Handler induction: action-indexed Markov operators	11
5.4	A tiny example	12
5.5	The left adjoint as lowering	12

6	Interlude: what $\{T_{\text{Moore}}, \text{rand}\}$ has taught us	12
6.1	Interface, implementation, semantics	12
6.2	Controlled Markov systems appear on the nose	13
6.3	Composition: implementation through implementation	13
6.4	The adjunction as a correctness principle	14
7	Replace probability by branching: $R = \text{List}_+$	14
7.1	Free branching	14
7.2	Reader equations kill the branching again	14
7.3	Moore through nonempty lists	15
7.4	May and must semantics from the same runner	15
7.5	Min and max	16
7.6	Order and multiplicity really are part of List_+	16
7.7	The left adjoint	16
8	A residual machine theory: Moore through store	16
8.1	Implementing Moore steps	17
8.2	Implementing get: a genuine residual read survives	17
8.3	The case with no private runner state	18
8.4	Handler induction	18
9	Reverse the direction: store through Moore	19
9.1	Classification theorem	19
9.2	A counting Moore machine	19
9.3	The step-detection lemma	20
9.4	Reduction to an O -indexed family of stores	21
9.5	Why the obvious register interpretation fails	21
9.6	The required register theory is Store itself	22
10	The asymmetry is the point	22
11	What Part II has added	23
12	Where to go next: the tight direction	24
	References and orientation	24

1 Where Part I left us

A residual stateful runner for a monad T in a residual monad R , with runner state S , is a family

$$\theta_X : TX \times S \longrightarrow R(X \times S),$$

natural in X and compatible with the two monad structures. Equivalently it is a monad morphism

$$T \longrightarrow \text{St}_S^R, \quad \text{St}_S^R(X) = R(X \times S)^S.$$

We will continue to write

$$T \rightsquigarrow R$$

for this loose, stateful direction.

Part I mostly took $R = \text{Id}$. Then the runner itself completely serviced the abstract operation:

$$TX \times S \longrightarrow X \times S.$$

For Moore theory this gave the familiar data

$$\delta : A \times S \rightarrow S, \quad o : S \rightarrow O.$$

There is a useful polynomial dictionary hidden in this familiar description. The Moore theory

$$T_{\text{Moore}}(A, O)$$

is generated by unary operations step_a , one for each $a \in A$, together with the O -ary reader operation get , subject only to the reader equations. A comodel therefore begins with cooperations

$$\text{step}_a : S \rightarrow S, \quad \text{get}_S : S \rightarrow O \times S.$$

The reader equations force the state component of get_S to be the identity, so uniquely

$$\text{get}_S(s) = (o(s), s)$$

for a map $o : S \rightarrow O$. The step cooperations assemble into

$$\delta : S \rightarrow S^A.$$

Thus a Moore comodel is exactly a coalgebra

$$S \longrightarrow O \times S^A$$

for the polynomial endofunctor

$$Oy^A.$$

In other words,

$$\boxed{\mathbf{Comod}(T_{\text{Moore}}(A, O)) \cong \mathbf{Coalg}(Oy^A).}$$

This dictionary will matter later: the polynomial records the *shape* of the machine interface, whereas the algebraic theory can impose additional equations on machines of that same shape.

Part II begins when the runner is allowed to leave something behind for a residual theory to do:

$$TX \times S \longrightarrow R(X \times S).$$

This is where the word *residual* becomes operationally meaningful.

1.1 Keep the handler direction straight

A runner Y with carrier S induces a functor

$$\Theta_Y : \mathbf{Alg}(R) \longrightarrow \mathbf{Alg}(T)$$

whose underlying carrier functor is

$$C \longmapsto C^S.$$

Thus an R -handler supplies a semantics for the residual effect, and the runner promotes it to a T -handler on state-indexed values.

In the locally presentable Set-based examples we use below, Θ_Y has a left adjoint

$$L_Y \dashv \Theta_Y.$$

The left adjoint may be read as *execution*, *lowering*, or *compilation*: it freely imposes the equations saying that an abstract T -operation is implemented by the residual R -computation specified by the runner.

The direction is worth fixing once and for all:

$$L_Y : \mathbf{Alg}(T) \rightleftharpoons \mathbf{Alg}(R) : \Theta_Y, \quad L_Y \dashv \Theta_Y.$$

The runner-induced handler functor is the *right* adjoint in this pair. Its underlying carrier is $(-)^S$. There is generally no further adjoint on its right.

1.2 Residual theory as machine language

Before turning to probability and other residual effects, it is worth pausing over a more familiar interpretation of the second theory R . A residual theory need not be thought of merely as an “effect which remains to be handled”. It may instead describe the *operational language provided by a substrate*. For example, suppose that R_{mach} is a theory describing the primitive instructions of some simple machine. Depending on the chosen formalism, these might include operations for reading and writing tape symbols, moving a head, manipulating registers or a stack, branching, and so forth. An actual machine which executes this instruction language is then an R_{mach} -comodel, or equivalently an

$$R_{\text{mach}} \rightsquigarrow \text{Id}$$

runner. Its state consists of the concrete machine configuration: tape contents, head position, control state, registers, or whatever other information the machine retains. Now let T be some higher-level theory. A runner

$$T \rightsquigarrow R_{\text{mach}}$$

says that operations of T can be realised by computations in the machine instruction language, while carrying some additional hidden implementation state. Composing with the machine itself gives

$$T \rightsquigarrow R_{\text{mach}} \rightsquigarrow \text{Id}.$$

This has the familiar reading

$$\text{source computation} \longrightarrow \text{machine-level computation} \longrightarrow \text{concrete execution}.$$

This example also makes the distinction between tight and loose maps particularly vivid. If every operation of T can simply be replaced by a fixed term of R_{mach} , with no additional state needed, then we may have an ordinary theory interpretation

$$T \longrightarrow R_{\text{mach}}.$$

This is analogous to a stateless compilation or macro-expansion. A loose map

$$T \rightsquigarrow R_{\text{mach}},$$

by contrast, can carry hidden state. It is therefore closer to a stateful interpreter or runtime implementation: the implementation may maintain a stack, heap, cache, register allocation, buffering state, or some other structure which is not visible in the source theory. This suggests a useful general interpretation of the residual theory:

$$\boxed{R = \text{the operational language supplied by the substrate.}}$$

In different examples the substrate may supply randomness, mutable store, nondeterministic choice, machine instructions, network operations, or some other collection of capabilities. A runner

$$T \rightsquigarrow R$$

then says that the T -interface is implemented using those capabilities. The idea becomes especially suggestive when R_{mach} is associated with a universal machine. Universality is not, strictly speaking, a property of the instruction theory alone: it also involves a choice of coding of programs and a machine capable of interpreting those codes. Nevertheless, schematically one may think of a universal implementation language R_U through which many computational theories factor:

$$T \rightsquigarrow R_U \rightsquigarrow \text{Id.}$$

For a sufficiently expressive class of source theories T , the same residual theory R_U can therefore serve as a common implementation target. From the point of view of $\mathbf{Run}(\mathcal{D})$, universality begins to resemble a factorisation property: many theories admit runners into one common operational substrate. One can continue the factorisation through several implementation levels:

$$T_{\text{high}} \rightsquigarrow T_{\text{VM}} \rightsquigarrow T_{\text{mach}} \rightsquigarrow \text{Id.}$$

Here one may read the successive theories as a high-level effect language, a virtual-machine or bytecode language, a low-level machine language, and finally concrete state evolution. Horizontal composition of runners is then literally *implementation through implementation*. The hidden state of the composite records whatever state is needed at all the intermediate levels. This example reinforces a point which will recur throughout the tutorial. The distinction between “abstract program” and “concrete machine” is relative rather than absolute. A residual theory R may be concrete relative to T , while itself being abstract relative to some further theory P . Thus

$$T \rightsquigarrow R \rightsquigarrow P$$

can be read as a hierarchy of embodiments, each layer realising the one above it. There is a technical caveat which we shall leave aside here. A fully satisfactory theory of universal Turing computation generally involves partiality, iteration or recursion, so an ordinary finitary Lawvere theory may not by itself be the right formal setting. One might instead use an iterative or Elgot-style theory, a partiality monad, or some other algebraic account of recursive computation. Nothing in the present example depends on choosing among these possibilities. Its purpose is simply to show that the residual slot R has a very ordinary computational interpretation: it can be the language in which another theory is implemented.

2 Our first residual effect: $T = \mathbb{N}y$, $R = \text{rand}$

We write rand for the polynomial lottery monad often denoted lott :

$$\text{rand}(X) = \coprod_{n \in \mathbb{N}} \coprod_{P \in \Delta_n} X^n.$$

An element remembers a finite set of lottery tickets, a probability distribution on those tickets, and the return attached to each ticket. This is deliberately a little richer than the ordinary finite-distribution monad: the lottery presentation itself is retained.

Take

$$T(X) = \mathbb{N} \times X,$$

the free monad on one unary operation. Since T is free, a runner is completely determined by what one abstract tick does:

$$\boxed{\alpha : S \longrightarrow \mathbf{rand}(S).}$$

At state s , one abstract tick is implemented by a lottery over next runner states.

If

$$\alpha(s) = (N_s, P_s, k_s), \quad k_s : N_s \rightarrow S,$$

then ticket $n \in N_s$ has probability $P_s(n)$ and sends the runner to $k_s(n)$.

The deterministic case is recovered by a one-ticket lottery:

$$\alpha(s) = \eta(f(s)).$$

So the comodel $f : S \rightarrow S$ from Part I embeds exactly as it should.

2.1 Several abstract ticks

Let $\alpha^{[0]} = \eta_S$, and define the Kleisli iterates recursively by

$$\alpha^{[n+1]}(s) = \alpha(s) \gg \alpha^{[n]}.$$

Then

$$\alpha^{[n]} : S \rightarrow \mathbf{rand}(S)$$

is the lottery of implementation states after n abstract ticks. The full runner is

$$\theta_X : \mathbb{N} \times X \times S \rightarrow \mathbf{rand}(X \times S),$$

with x carried unchanged while the runner state evolves stochastically.

The monad law says exactly what we want operationally: implementing $m + n$ abstract ticks is the same as implementing m ticks and then n more, with lottery substitution providing the concrete sequencing law.

2.2 A residual handler turns the lottery into a value

Let

$$\rho : \mathbf{rand}(C) \rightarrow C$$

be a $\mathbf{rand}$ -algebra. The runner induces a unary T -operation on C^S . For $h : S \rightarrow C$,

$$\boxed{(Uh)(s) = \rho(\mathbf{rand}(h)(\alpha(s))).}$$

The formula decomposes into three steps:

$$S \xrightarrow{\alpha} \mathbf{rand}(S) \xrightarrow{\mathbf{rand}(h)} \mathbf{rand}(C) \xrightarrow{\rho} C.$$

So the runner produces a residual lottery and the handler says what that lottery means.

2.3 Expectation: a Markov operator appears

Take $C = \mathbb{R}$ and let ρ evaluate a lottery by expectation:

$$\rho(N, P, r) = \sum_{n \in N} P(n)r(n).$$

Then

$$(Uh)(s) = \sum_{n \in N_s} P_s(n)h(k_s(n)).$$

Equivalently,

$$(Uh)(s) = \mathbb{E}[h(S_{t+1}) \mid S_t = s].$$

Thus the first genuinely residual Sweedler runner has already produced the familiar backward Markov operator on observables.

2.4 Entropy plus continuation cost

There is another useful **rand**-algebra on $[0, \infty)$:

$$\rho(N, P, r) = H(P) + \sum_n P(n)r(n),$$

where

$$H(P) = - \sum_n P(n) \log P(n).$$

The algebra law is Shannon's chain-rule phenomenon: evaluating a compound lottery in stages agrees with flattening it first.

For this handler,

$$(Uh)(s) = H(P_s) + \sum_n P_s(n)h(k_s(n)).$$

So one abstract tick receives the semantics

immediate informational cost + expected continuation cost.

The runner itself has not changed. Only the residual handler has.

This is our first major lesson of Part II:

$$\text{runner} + \text{interpretation of the residual effect} \implies \text{higher-level handler.}$$

The runner says how a T -operation produces an R -effect. An R -algebra says how that effect is evaluated.

2.5 The left adjoint in this example

Let (B, g) be a **Ny**-handler. The left adjoint $L_Y(B)$ is the **rand**-algebra freely generated from formal pairs (b, s) , subject to the execution rule

$$\eta(g(b), s) = \text{rand}(s' \mapsto (b, s'))(\alpha(s)).$$

In Part I the corresponding relation was simply

$$(g(b), s) \sim (b, f(s)).$$

Probability has replaced one concrete successor by a residual lottery of successors.

3 Free branching through probability: $F(y^O) \rightsquigarrow \mathbf{rand}$

Now let

$$T = F(y^O),$$

the free monad on one O -ary operation. Since the theory is free, a runner is again determined by its action on the generator:

$$\boxed{\alpha : S \longrightarrow \mathbf{rand}(O \times S).}$$

At state s , one abstract branching operation is realised by a lottery over pairs

$$(o, s') \in O \times S.$$

The lottery may therefore decide both

which continuation is taken and what the next implementation state is.

These two random choices may be correlated.

3.1 Running a whole tree

An element of $F(y^O)(X)$ is a finite O -branching tree with leaves in X . The runner extends recursively to

$$\theta_X : F(y^O)(X) \times S \rightarrow \mathbf{rand}(X \times S).$$

At a leaf,

$$\theta_X(x, s) = \eta(x, s).$$

At a branching node $(t_o)_{o \in O}$,

$$\boxed{\theta_X((t_o)_{o \in O}, s) = \alpha(s) \gg \! \! \! \times ((o, s') \mapsto \theta_X(t_o, s')).}$$

Thus there is now a lottery of execution paths through the tree.

3.2 The induced O -ary handler operation

Let $\rho : \mathbf{rand}(C) \rightarrow C$ be a residual handler and take an O -indexed family

$$h_o : S \rightarrow C.$$

Define

$$q_h : O \times S \rightarrow C, \quad q_h(o, s') = h_o(s').$$

Then the induced O -ary operation on C^S is

$$\boxed{\beta((h_o)_{o \in O})(s) = \rho(\mathbf{rand}(q_h)(\alpha(s))).}$$

With expectation underneath, this becomes

$$\beta((h_o))(s) = \sum_{n \in N_s} P_s(n) h_{o_n}(s_n),$$

where ticket n returns (o_n, s_n) .

With the entropy algebra,

$$\beta((h_o))(s) = H(P_s) + \sum_n P_s(n) h_{o_n}(s_n).$$

So the abstract branch node acquires an informational stage cost plus expected continuation cost.

3.3 The lottery presentation can matter

Two different tickets may return the same pair (o, s') . The ordinary distribution monad would combine their probabilities, but **rand** retains the ticket presentation. Expectation forgets this distinction after evaluation. The entropy algebra need not: its immediate term is the entropy of the implementation lottery itself.

This is a useful reminder that the residual theory records an *implementation shape*; a particular handler may later forget some of that shape.

4 Quotient to reader: the equations kill the randomness

Now quotient the free branching theory to the reader monad

$$T(X) = X^O.$$

The reader equations may be written

$$\mathbf{get}(o \mapsto x) = x$$

and

$$\mathbf{get}(o \mapsto \mathbf{get}(o' \mapsto x_{oo'})) = \mathbf{get}(o \mapsto x_{oo}).$$

Start from the free runner datum

$$\alpha : S \rightarrow \mathbf{rand}(O \times S).$$

Which such runners descend to the quotient?

4.1 The first reader equation is already decisive

Run

$$\mathbf{get}(o \mapsto x)$$

from state s . The branch label is discarded, so the left side produces the lottery obtained by mapping

$$(o, s') \mapsto (x, s').$$

The right side performs no operation, hence gives the unit lottery $\eta(x, s)$.

Since **rand** remembers its ticket presentation, equality forces the original lottery to have exactly the unit-lottery shape. Therefore, for a unique $o(s) \in O$,

$$\boxed{\alpha(s) = \eta(o(s), s).}$$

So the entire runner collapses to

$$\boxed{o : S \rightarrow O.}$$

The residual randomness disappears.

The second reader equation is then automatic: reading twice from the same unchanged state selects $o(s)$ twice.

This corrects a tempting but wrong implementation story. One might try to “sample an o once and remember it”. But the equation $\mathbf{get}(o \mapsto x) = x$ says that a discarded read must be indistinguishable from doing nothing, including at the runner-state and residual-effect level. Hidden memoisation therefore violates the theory.

4.2 The phenomenon is not only about ticket sensitivity

For the ordinary finite-distribution monad, the first reader equation would still force the implementation state to remain fixed, but it could leave a distribution q_s on O . The second reader equation compares two independent reads with one diagonal read; equality for arbitrary continuations forces q_s to be a point mass. So the reader equations kill genuine random reading there as well.

4.3 Handler induction becomes independent of the residual algebra

For a **rand**-algebra $\rho : \mathbf{rand}(C) \rightarrow C$, our free-branching formula now reduces to

$$\begin{aligned}\beta((h_o))(s) &= \rho(\eta(h_{o(s)}(s))) \\ &= h_{o(s)}(s).\end{aligned}$$

Thus

$$\boxed{\beta((h_o))(s) = h_{o(s)}(s),}$$

exactly as in Part I. Expectation, entropy, or any other residual handler all induce the same abstract read.

Quotient equations in the source theory are *implementation constraints*. They must already hold at the residual-effect level; it is not enough that some later handler happens to erase the difference.

5 Moore machines through probability

Let T_{Moore} be generated by

$$Ay + y^O,$$

with reader equations on **get** and no mixed equations relating **step_a** to **get**.

For each input $a \in A$, the unary operation may be realised by an arbitrary lottery over next implementation states:

$$\delta_a : S \rightarrow \mathbf{rand}(S).$$

Together,

$$\boxed{\delta : A \times S \rightarrow \mathbf{rand}(S).}$$

The reader calculation forces the observation part to remain deterministic and state-preserving:

$$\boxed{o : S \rightarrow O.}$$

Hence

$$\boxed{\{T_{\text{Moore}}, \mathbf{rand}\}\text{-runner} = (S, \delta : A \times S \rightarrow \mathbf{rand}(S), o : S \rightarrow O).}$$

There is no additional compatibility equation between δ and o , because the source Moore theory imposed none.

5.1 Running a whole Moore computation

The full interpreter is

$$\theta_X : T_{\text{Moore}}X \times S \rightarrow \text{rand}(X \times S).$$

At a return value,

$$\theta_X(x, s) = \eta(x, s).$$

At an input step,

$$\theta_X(\text{step}_a(t), s) = \delta(a, s) \gg= (s' \mapsto \theta_X(t, s')).$$

At get,

$$\theta_X(\text{get}((t_o)_{o \in O}), s) = \theta_X(t_{o(s)}, s).$$

So step may create a lottery, while get itself never does.

5.2 The output process can still be random

Although $o : S \rightarrow O$ is deterministic, after a stochastic transition the next observed output is random. Starting from s under input a , the output lottery is simply the pushforward

$$\text{rand}(o)(\delta(a, s)) \in \text{rand}(O).$$

The theory has therefore not made the overall behaviour deterministic. It has said something subtler: randomness enters through state evolution, while observation faithfully reads the state actually reached.

5.3 Handler induction: action-indexed Markov operators

Let $\rho : \text{rand}(C) \rightarrow C$ be a residual handler. The induced Moore handler on C^S has transition operations

$$(\text{step}_a h)(s) = \rho(\text{rand}(h)(\delta(a, s)))$$

and reader operation

$$\text{get}((h_i)_{i \in O})(s) = h_{o(s)}(s).$$

With expectation on $C = \mathbb{R}$, every input a gives a backward Markov operator

$$U_a : \mathbb{R}^S \rightarrow \mathbb{R}^S,$$

$$(U_a h)(s) = \mathbb{E}[h(S') \mid a, s].$$

With the entropy algebra,

$$(U_a h)(s) = H(\delta(a, s)) + \mathbb{E}[h(S') \mid a, s],$$

where the entropy is that of the implementation lottery presentation at (a, s) .

5.4 A tiny example

Let

$$A = \{L, R\}, \quad S = \{0, 1, 2\}, \quad O = \{\text{red}, \text{blue}\},$$

and suppose

$$o(0) = o(1) = \text{red}, \quad o(2) = \text{blue}.$$

If

$$\delta(L, 0) = \frac{1}{2}[0] + \frac{1}{2}[2],$$

then after input L from state 0, the next output is red with probability 1/2 and blue with probability 1/2. Yet a get performed at any actual state is completely deterministic.

5.5 The left adjoint as lowering

Let B be a Moore handler, with unary operations $u_a : B \rightarrow B$ and reader operation $g : B^O \rightarrow B$. The left adjoint freely generates a **rand**-algebra from pairs (b, s) , imposing

$$\boxed{\eta(u_a(b), s) = \text{rand}(s' \mapsto (b, s'))(\delta(a, s))}$$

and

$$\boxed{\eta(g((b_i)), s) = \eta(b_{o(s)}, s)}.$$

The abstract Moore step is compiled to the residual lottery prescribed by the implementation. The abstract read compiles away without residual randomness.

6 Interlude: what $\{T_{\text{Moore}}, \text{rand}\}$ has taught us

This example is worth pausing over because it is the first one that looks like a useful systems formalism in its own right.

6.1 Interface, implementation, semantics

There are now three genuinely separate layers:

$$\boxed{\begin{array}{c} \text{abstract theory } T \\ \downarrow \\ \text{runner } Y : T \rightsquigarrow R \\ \downarrow \\ \text{residual handler } C \in \mathbf{Alg}(R). \end{array}}$$

They can be read as

$$\text{interface} \quad + \quad \text{implementation} \quad + \quad \text{semantic evaluation}.$$

The same stochastic runner may be evaluated by expectation, entropy-plus-expectation, or another **rand**-algebra without changing the implementation architecture itself.

6.2 Controlled Markov systems appear on the nose

The data

$$\delta : A \times S \rightarrow \text{rand}(S), \quad o : S \rightarrow O$$

are an action-indexed stochastic transition system with deterministic state labelling. If a sequence of inputs $a_0, a_1, \dots$ is supplied, then

$$S_{t+1} \sim \delta(a_t, S_t), \quad O_t = o(S_t).$$

A controller $\pi : S \rightarrow A$, or an observation-based controller $\pi : O \rightarrow A$, closes the open controlled system to an ordinary stochastic process.

Sweedler theory adds a different emphasis: this stochastic system is not merely the machine we decided to study. It is the *implementation data* of the deterministic Moore theory in a probabilistic residual theory.

Because the Moore **get-cooperation** is deterministic and idempotent in the reader sense, a POMDP with genuinely stochastic observations is not directly a

$$T_{\text{Moore}} \rightsquigarrow \text{rand}$$

runner: the observation randomness must either be absorbed into an enlarged hidden state, or represented by a Mealy-style observation mechanism in which output is generated during the transition. This is the same E_1 -idempotence obstruction that will reappear in the later POMDP example.

6.3 Composition: implementation through implementation

Suppose

$$Y : T \rightsquigarrow R$$

has runner state S , and

$$Z : R \rightsquigarrow Q$$

has runner state U . Their composite has state $S \times U$. Concretely,

$$\begin{aligned} TX \times (S \times U) &\cong (TX \times S) \times U \\ &\xrightarrow{\theta_X^Y \times 1} R(X \times S) \times U \\ &\xrightarrow{\theta_{X \times S}^Z} Q((X \times S) \times U) \\ &\cong Q(X \times (S \times U)). \end{aligned}$$

So implementation layers stack, and their private states accumulate as a product.

The handler functors compose in exactly the corresponding way:

$$\mathbf{Alg}(Q) \xrightarrow{\Theta_Z} \mathbf{Alg}(R) \xrightarrow{\Theta_Y} \mathbf{Alg}(T).$$

On carriers,

$$(C^U)^S \cong C^{S \times U}.$$

This is a precise version of the slogan that one may realise a high-level machine through an intermediate implementation theory and then realise that intermediate theory again.

6.4 The adjunction as a correctness principle

The adjunction

$$L_Y \dashv \Theta_Y$$

says

$$\mathbf{Alg}(R)(L_Y B, C) \cong \mathbf{Alg}(T)(B, \Theta_Y C).$$

The left side says: lower the abstract handler B to the residual theory, then interpret it in C . The right side says: interpret B directly using the runner-induced T -handler on C^S .

It is therefore natural to read the adjunction as a universal correctness statement:

$$\text{interpret after lowering} = \text{interpret abstractly through the runner.}$$

7 Replace probability by branching: $R = \text{List}_+$

Now keep the same abstract theory but swap the residual effect. Let

$$\text{List}_+(X) = \coprod_{n \geq 1} X^n$$

be the nonempty-list monad. Its multiplication concatenates lists. An Eilenberg–Moore algebra for List_+ is exactly a semigroup.

This makes a useful comparison with `rand`:

<code>rand</code>		evaluate lotteries
List_+		combine nonempty finite sequences.

Because lists remember order and multiplicity, List_+ is richer than extensional nondeterminism. We will exploit that rather than hide it.

7.1 Free branching

For $T = F(y^O)$, a runner is simply

$$\alpha : S \rightarrow \text{List}_+(O \times S).$$

At state s , an abstract branching operation may be realised by a nonempty list

$$[(o_1, s_1), \dots, (o_n, s_n)].$$

Running a whole tree follows each listed implementation branch and concatenates the resulting lists of outcomes.

7.2 Reader equations kill the branching again

For reader theory, the equation

$$\text{get}(o \mapsto x) = x$$

turns the implementation list into

$$[(x, s_1), \dots, (x, s_n)]$$

on the left and

$$[(x, s)]$$

on the right. Equality of lists forces

$$n = 1, \quad s_1 = s.$$

Hence

$$\boxed{\{y^O, \text{List}_+\}\text{-runner} \cong (S \xrightarrow{o} O).}$$

Once again the reader equations leave no room for a residual branch at get.

7.3 Moore through nonempty lists

For Moore theory we therefore obtain

$$\boxed{\{T_{\text{Moore}}, \text{List}_+\}\text{-runner} = (S, \delta : A \times S \rightarrow \text{List}_+(S), o : S \rightarrow O).}$$

An abstract action may fork execution into a finite nonempty sequence of implementation states, while get remains deterministic.

If

$$\delta(a, s) = [s_1, \dots, s_n],$$

and $(C, *)$ is a semigroup, then the induced transition on C^S is

$$\boxed{(\text{step}_a h)(s) = h(s_1) * \dots * h(s_n).}$$

The get operation is unchanged:

$$\text{get}((h_i))(s) = h_{o(s)}(s).$$

7.4 May and must semantics from the same runner

Take $C = 2$.

With semigroup operation $\vee$,

$$(\text{step}_a h)(s) = \bigvee_{s' \in \delta(a, s)} h(s'),$$

so the predicate holds when *some* listed successor satisfies it. This is a may or existential reading.

With semigroup operation $\wedge$,

$$(\text{step}_a h)(s) = \bigwedge_{s' \in \delta(a, s)} h(s'),$$

so the predicate holds only when *every* listed successor satisfies it. This is a must or universal reading.

Thus

$$\boxed{\text{same operational branching} + \text{different residual algebra} = \text{different modal semantics}.}$$

7.5 Min and max

On a suitable ordered numerical carrier, choose

$$x * y = \min(x, y)$$

or

$$x * y = \max(x, y).$$

Then the same branching runner yields best-case or worst-case propagation of values. This is the nondeterministic analogue of choosing expectation versus another evaluator in the probabilistic case.

7.6 Order and multiplicity really are part of List_+

The transitions

$$[s_1, s_2], \quad [s_2, s_1], \quad [s_1, s_1, s_2]$$

are distinct at the residual-theory level. A commutative idempotent handler such as $(2, \vee)$ forgets those distinctions, but a general semigroup need not. Strings under concatenation, for example, remember both order and repetition.

If we wanted extensional finite nondeterminism from the start, we could instead use the finite nonempty powerset monad. Its algebras are semilattices. For this tutorial, however, List_+ is useful precisely because it stays polynomial and keeps the implementation mechanics visible.

7.7 The left adjoint

Let B be a Moore handler. The left adjoint freely generates a semigroup from pairs (b, s) , subject to

$$\boxed{[(u_a(b), s)] = [(b, s_1), \dots, (b, s_n)]}$$

whenever

$$\delta(a, s) = [s_1, \dots, s_n],$$

and

$$\boxed{[(g((b_i)), s)] = [(b_{o(s)}, s)]}.$$

The probabilistic implementation equation has simply been replaced by a list-shaped implementation equation.

8 A residual machine theory: Moore through store

Probability and lists contributed generic branching shapes. We now let the residual theory itself carry a structured machine capability.

Fix a store-value set V , and let

$$T_{\text{store}}(X) = V \rightarrow (X \times V)$$

be the global-state monad. A residual store computation may inspect the current store value and return both an ordinary result and a new store value.

Before considering runners, notice that store and Moore already have the same underlying operation signature. Store is presented by a V -ary lookup operation together with unary updates put_v , so its signature is

$$y^V + Vy.$$

This is the same signature as $T_{\text{Moore}}(V, V)$. On the comodel side, both therefore have the same polynomial interface

$$Vy^V,$$

that is, underlying data of the form

$$S \longrightarrow V \times S^V.$$

The difference is equational. An arbitrary Vy^V -coalgebra is a $T_{\text{Moore}}(V, V)$ -comodel, whereas a store comodel must in addition satisfy the familiar lookup–update equations. Equivalently, there is a tight quotient of theories

$$\boxed{T_{\text{Moore}}(V, V) \twoheadrightarrow T_{\text{store}}}$$

sending

$$\text{step}_v \longmapsto \text{put}_v, \quad \text{get} \longmapsto \text{lookup}.$$

Thus Moore and store give a particularly clean instance of the distinction

$$\boxed{\text{same polynomial interface} \not\Rightarrow \text{same theory.}}$$

The polynomial remembers the shape Vy^V ; the theory remembers which equations govern the use of that interface.

Now take

$$\{T_{\text{Moore}}, T_{\text{store}}\}.$$

A runner has private state S , in addition to the residual store state V . It is often helpful to think of the total concrete state as

$$S \times V.$$

8.1 Implementing Moore steps

For each input $a \in A$, the unary generator may be implemented by an arbitrary state computation

$$\delta_a : S \rightarrow T_{\text{store}}(S).$$

Equivalently,

$$\boxed{\bar{\delta} : A \times S \times V \rightarrow S \times V.}$$

Thus one abstract Moore input may inspect and modify both the runner’s private state and the residual store.

8.2 Implementing get: a genuine residual read survives

Before imposing the reader equations, get would give

$$\gamma : S \rightarrow T_{\text{store}}(O \times S).$$

Write

$$\gamma(s)(v) = ((o', s'), v').$$

The equation

$$\text{get}(o \mapsto x) = x$$

forces

$$s' = s, \quad v' = v,$$

but it does *not* force o' to be independent of the store value. Hence the lawful form is

$$\gamma(s)(v) = ((o(s, v), s), v)$$

for a map

$$o : S \times V \rightarrow O.$$

The second reader equation is then automatic.

So

$$\{T_{\text{Moore}}, T_{\text{store}}\}\text{-runner} = (S, \bar{\delta} : A \times S \times V \rightarrow S \times V, o : S \times V \rightarrow O).$$

This is our first example where the residual theory directly services an abstract read. The reader equations prohibit changing the world while reading it, but they do not prohibit *depending* on the world being read.

Compare the three residual theories at get:

rand	genuine random read is forbidden
List₊	genuine branching read is forbidden
T_{store}	a genuine store lookup is allowed, provided it is non-mutating.

The equations of the source theory interact with the specific capabilities of the residual theory, not just with an undifferentiated notion of “effect”.

8.3 The case with no private runner state

If $S = 1$, the data reduce to

$$\delta : A \times V \rightarrow V, \quad o : V \rightarrow O.$$

This is simply a Moore machine whose state is the store itself. Input a updates the store value; output is read from it. Here “Moore machine realised by store” is completely literal.

8.4 Handler induction

Let

$$\rho : T_{\text{store}}(C) \rightarrow C$$

be a store algebra. If

$$\bar{\delta}(a, s, v) = (s', v'),$$

then the induced transition on C^S is

$$(\text{step}_a h)(s) = \rho(v \mapsto (h(s'), v')).$$

For get,

$$\text{get}((h_i))(s) = \rho(v \mapsto (h_{o(s, v)}(s), v)).$$

The residual store is genuinely consulted to choose the abstract output branch, while the store itself is left unchanged by that observation.

9 Reverse the direction: store through Moore

Now consider

$$\boxed{\{T_{\text{store}}, T_{\text{Moore}}\}}.$$

This asks a much stronger question: can the equations of mutable store be implemented using the generic operations of Moore theory?

Present store algebraically by a V -ary lookup operation and unary updates put_v , subject to the usual global-state equations: repeated lookup is coherent, reading after writing v returns v , last write wins, and reading then restoring the value changes nothing. A runner with private state S must implement every source operation by a Moore computation. Thus for each $v \in V$ we need

$$p_v : S \rightarrow T_{\text{Moore}}S,$$

while lookup requires

$$\ell : S \rightarrow T_{\text{Moore}}(V \times S).$$

The store equations must hold as equalities of Moore computations.

At first sight there seem to be many possibilities: the runner might keep the store entirely in its private state, or it might try to exploit the Moore transition operations. In fact the store equations rule out the second possibility completely.

9.1 Classification theorem

For fixed A and O , a runner

$$T_{\text{store}} \rightsquigarrow T_{\text{Moore}}(A, O)$$

with private state S is equivalently an O -indexed family of V -store comodel structures on S . Equivalently, in **Set**, it is an O -indexed family of decompositions

$$S \cong V \times K_O.$$

In particular, the classification is independent of A .

The point is stronger than the observation that one may ignore the Moore target. Every lawful runner is forced to ignore its transition operations. The residual Moore theory can contribute only its read-only O -environment.

9.2 A counting Moore machine

To see why, consider the Moore machine with state

$$\mathbb{N} \times O^{\mathbb{N}}.$$

Every input increments the counter,

$$\delta_a(n, \omega) = (n + 1, \omega),$$

while observation reads the current entry of the stream,

$$o(n, \omega) = \omega(n).$$

Composing a putative runner

$$T_{\text{store}} \rightsquigarrow T_{\text{Moore}}(A, O)$$

with this Moore comodel gives a genuine store comodel on

$$S \times \mathbb{N} \times O^{\mathbb{N}}.$$

Thus every Moore **step** executed by the runner becomes visible as an increase of the $\mathbb{N}$ -coordinate.

Write $c_v(z) \in \mathbb{N}$ for the number of Moore steps executed by the implementation of put_v from a composite state z . The store equations include

$$\text{put}_{\text{lookup}(z)}(z) = z$$

and last-write-wins. Hence

$$c_{\text{lookup}(z)}(z) = 0,$$

and, after first writing w ,

$$c_w(z) + c_{\text{lookup}(z)}(\text{put}_w z) = c_{\text{lookup}(z)}(z) = 0.$$

Since both summands are natural numbers,

$$\boxed{c_w(z) = 0.}$$

Thus no update operation can execute a Moore step. The same conclusion holds for lookup, since lookup is state-preserving in a store comodel.

9.3 The step-detection lemma

The counting argument becomes a statement about terms of Moore theory.

If a term of $T_{\text{Moore}}(A, O)$ executes zero steps in the counting comodel

$$\mathbb{N} \times O^{\mathbb{N}}$$

for every $\omega \in O^{\mathbb{N}}$, then, modulo the reader equations, it is equal to a step-free term.

Indeed, put a Moore term into reader-normal form. Before the first **step**, any successive gets see the same observation, exactly as required by the reader equations. If a reachable first step remains, the arbitrary stream ω can be chosen so that the preceding gets select the branch leading to that step. The counter then increases, contradicting the hypothesis. Hence every branch reachable under some observation is step-free, and the term lies in the reader fragment.

There is therefore a split inclusion

$$y^O \longrightarrow T_{\text{Moore}}(A, O),$$

with retraction

$$T_{\text{Moore}}(A, O) \longrightarrow y^O$$

which sends every step_a to the identity computation. The classification above says that every store runner into Moore factors through this reader retract:

$$T_{\text{store}} \rightsquigarrow y^O \longrightarrow T_{\text{Moore}}(A, O).$$

This is the precise sense in which the answer is independent of the input set A .

9.4 Reduction to an O -indexed family of stores

For residual Reader, the state-transformer monad is

$$\begin{aligned} \text{St}_S^{y^O}(X) &= S \rightarrow y^O(X \times S) \\ &= S \rightarrow (X \times S)^O \\ &\cong O \rightarrow (S \rightarrow X \times S) \\ &= \prod_{o \in O} \text{St}_S(X). \end{aligned}$$

The monad structure is pointwise in o . Consequently a monad map

$$T_{\text{store}} \longrightarrow \text{St}_S^{y^O}$$

is exactly an O -indexed family of monad maps

$$T_{\text{store}} \longrightarrow \text{St}_S,$$

that is, an O -indexed family of V -store comodel structures on S .

In **Set**, a lawful store comodel is equivalently a decomposition

$$S \cong V \times K$$

for some complement K . Hence a runner into Moore amounts to choosing, for each $o \in O$, a decomposition

$$S \cong V \times K_o.$$

The observation supplied by the residual Moore theory acts only as the parameter selecting which store structure is in force.

The familiar “boring solution” is recovered as a special case. Taking

$$S = V$$

and choosing the same standard store structure for every $o \in O$ gives a constant O -indexed family. What looked before like one uninteresting escape route is therefore one point in the complete classification.

9.5 Why the obvious register interpretation fails

A tempting idea is still to take

$$A = V, \quad O = V,$$

interpret put_v by the Moore input v , and interpret lookup by Moore get . This sounds exactly like a register. The theorem explains precisely why it fails: every lawful Store-to-Moore runner factors through Reader, so no step_v may occur in the implementation of a store operation.

The generic Moore theory does not assert

$$\text{step}_v; \text{step}_w = \text{step}_w,$$

or that

$$\text{step}_v; \text{get}$$

must subsequently return v . An individual Moore machine may satisfy these properties, but they are not identities of Moore theory.

A generic Moore target can host a store only by treating its output as a fixed environment parameter selecting a store structure on the private runner state. It can never use its Moore transitions to implement a lawful write.

This makes the distinction between interface and equations especially sharp. Moore and Store expose the same polynomial interface Vy^V , but the Store equations force any runner into the generic Moore theory to ignore the very transition port which, at the level of Poly alone, looks like the natural candidate for writing.

9.6 The required register theory is Store itself

The failure also identifies exactly what would repair the target. Quotient $T_{\text{Moore}}(V, V)$ by the usual equations for writing and reading a single cell. The resulting “register theory” is precisely T_{store} . Thus there is a tight quotient

$$T_{\text{Moore}}(V, V) \twoheadrightarrow T_{\text{store}},$$

sending

$$\text{step}_v \longmapsto \text{put}_v, \quad \text{get} \longmapsto \text{lookup}.$$

Equivalently, this tight map may be viewed as a runner with trivial residual state. The target now guarantees the store equations because they are literally equations of the target theory.

Moore versus Store separates interface from law:

$$\begin{array}{ll} T_{\text{Moore}}(V, V) & : \text{ all } Vy^V\text{-coalgebras,} \\ T_{\text{store}} & : \text{ those } Vy^V\text{-coalgebras satisfying the store equations.} \end{array}$$

Poly records the shape of the interface; the theory records which uses of that interface are lawful.

10 The asymmetry is the point

The two Sweedler homs have very different characters.

For

$$\{T_{\text{Moore}}, T_{\text{store}}\},$$

the residual store offers abundant stateful capability. The source Moore theory asks only for arbitrary input transitions and a lawful read, so we obtain the concrete data

$$A \times S \times V \rightarrow S \times V, \quad S \times V \rightarrow O.$$

For

$$\{T_{\text{store}}, T_{\text{Moore}}\},$$

the source store theory brings strong equations that the generic Moore transition structure cannot satisfy. The classification theorem above shows exactly what remains: every runner factors through the Reader fragment, so the Moore output may parameterise a family of store structures on the private runner state, but the Moore input transitions cannot participate in a lawful implementation.

This asymmetry is not a defect. It is exactly what $\{T, R\}$ is supposed to detect:

ways of implementing the operations and equations of T using the capabilities of R .

11 What Part II has added

It is worth collecting the examples in one place.

Residual theory	Moore runner data	What the residual theory contributes
Id	$A \times S \rightarrow S, S \rightarrow O$	complete implementation by the runner itself
rand	$A \times S \rightarrow \text{rand}(S), S \rightarrow O$	probabilistic transition
List ₊	$A \times S \rightarrow \text{List}_+(S), S \rightarrow O$	finite sequential branching
T_{store}	$A \times S \times V \rightarrow S \times V, S \times V \rightarrow O$	mutable state and genuine residual lookup

Several themes have become stable enough to state explicitly.

1. **Equations in the source theory constrain implementations.** Reader equations killed random and list-valued reads, while permitting a pure store lookup.
2. **The runner and the residual handler play different roles.** The runner specifies how abstract operations produce residual computations; the handler evaluates those computations.
3. **Handler induction is contravariant in runner state.** The induced semantics lives on C^S : forward evolution of implementation state produces backward propagation of predicates, values, or continuations.
4. **The left adjoint is naturally read as lowering.** It freely converts abstract handler syntax into residual-effect syntax according to the execution equations of the runner.
5. **Runners compose.** Implementations may be layered, with state spaces multiplying and handler functors composing.
6. **Machine-on-machine realisation is asymmetric.** The operations and, especially, the equations of the target theory determine what can actually be implemented.
7. **Polynomial interface and theory are different levels of description.** Moore and store can expose the same interface polynomial Vy^V while differing by equations. The polynomial records the shape of observation and transition; the theory records the laws those operations must satisfy.

12 Where to go next: the tight direction

Everything in Parts I and II has concentrated on the loose, stateful direction

$$T \rightsquigarrow R.$$

There is another basic way theories relate: an ordinary monad or theory map

$$f : T \longrightarrow R.$$

Such a map carries no private runner state. It directly interprets every T -operation as an R -operation, compatibly with substitution and equations. On handlers it acts by restriction:

$$\mathbf{Alg}(R) \longrightarrow \mathbf{Alg}(T)$$

without changing the underlying carrier.

So Part III will begin from a sharp contrast:

$$\begin{array}{l|l} \text{tight interpretation } T \rightarrow R & C \mapsto C \\ \text{loose runner } T \rightsquigarrow R \text{ on } S & C \mapsto C^S. \end{array}$$

Maps such as

$$T \longrightarrow T_1 + \cdots + T_n$$

can often be read as wiring: the operations expected by a composite interface are interpreted using operations supplied by several component theories. But not every tight map is merely a wiring diagram; theory interpretations also include renaming, forgetting, compilation of primitives into composite terms, and refinement.

The natural next question is therefore how the two directions interact. In the double-category picture one expects squares of the form

$$\begin{array}{ccc} T & \rightsquigarrow & R \\ \downarrow & & \downarrow \\ T' & \rightsquigarrow & R', \end{array}$$

expressing compatibility between changing an interface tightly and changing its implementation loosely.

That will be the theme of Part III: interpretations, wiring, and the interaction of tight structure with runners.

References and orientation

The general runner facts used throughout are from:

- D. McDermott, E. Rivas, and T. Uustalu, *Sweedler Theory of Monads*, FoSSaCS 2022, LNCS 13242, 428–448. In particular, an R -residual stateful runner with carrier S is a family $TX \times S \rightarrow R(X \times S)$, and it induces a functor $\mathbf{Alg}(R) \rightarrow \mathbf{Alg}(T)$ carried by the internal $\text{hom } S \Rightarrow -$.
- T. Uustalu, *Stateful Runners of Effectful Computations*, Electronic Notes in Theoretical Computer Science 319 (2015), 403–421, for the algebraic-effects and comodel viewpoint on stateful runners.

For the polynomial lottery monad used here, see D. Spivak, *Lotteries: a constructive version of the distributions monad*, Topos Institute blog, 23 March 2023. The monad called **rand** in this tutorial is the polynomial lottery monad often denoted **lott**; it retains the lottery-ticket presentation rather than quotienting immediately to a finitely supported distribution.

The expectation and entropy examples use two algebra structures on this lottery monad. The entropy-plus-expectation operation

$$(N, P, r) \mapsto H(P) + \sum_n P(n)r(n)$$

is compatible with substitution by the Shannon chain rule, which is why it behaves as a lawful handler for compound lotteries.

The picture to retain from Part II is

$$\boxed{\text{abstract theory} \xrightarrow{\text{stateful realisation}} \text{residual theory} \xrightarrow{\text{handler}} \text{semantics.}}$$

Part III will add the other axis: direct, tight interpretations between theories, including the maps that can serve as wiring diagrams.