

Getting to Grips with Sweedler Runners for Systems

Introduction: systems as stateful realisations between theories
A conversational guide to the tutorial and the research programme behind it

October 2026

What this tutorial is about. A great many categorical descriptions of systems begin either with a state space, an interface, or a wiring diagram. This tutorial explores a complementary starting point: begin with *theories of interaction*, and ask how one such theory can be realised inside another by a stateful mechanism. Sweedler theory supplies a universal object $\mathcal{M}(T, R)$ whose coalgebras are precisely such mechanisms in the main endofunctor example. We call them *runners*. The central systems-theoretic proposal is to organise theories, direct interpretations, stateful runners, and transformations of runners into a double-categorical architecture $\mathbf{Run}(\mathcal{D})$.

The series is intentionally exploratory. Some ingredients are established theorems; others are a proposed way of organising them for categorical systems theory; still others are research questions that the examples force upon us. The aim is to learn the architecture by using it.

Contents

1	Why another categorical language for systems?	2
2	The basic formula	2
3	A first familiar machine appears	3
4	The Sweedler hom packages all runners at once	3
5	The proposed architecture: $\mathbf{Run}(\mathcal{D})$	4
6	Runners also act on ways of using a system	5
7	Why probability and nondeterminism are revealing	5
8	The ambient category is part of the modelling choice	6
9	A second axis: theories can vary too	7
10	What the tutorial is trying to teach	7
11	A provisional map of the series	8

12 Established mathematics and proposed architecture	9
12.1 Established ingredients	9
12.2 The organising proposal	9
12.3 Research questions	9
13 A philosophical reading, if one wants one	10
14 How to read the tutorial	10

1 Why another categorical language for systems?

Categorical systems theory already has many successful languages. A state machine can be described as a coalgebra. An open dynamical system can be described by a lens-like construction. Wiring diagrams tell us how components are interconnected. Algebraic theories and monads specify computational effects. Sheaf and presheaf semantics organise behaviour over time or over contexts. Petri nets and higher-dimensional automata expose concurrency and resource flow. None of these descriptions is dispensable.

The difficulty appears when a single system participates in several of these roles at once. A component may have a hidden state, present one interface externally, implement that interface in terms of another computational language internally, leave some effects unresolved for a downstream environment, and carry logical or quantitative guarantees about what happens as it runs. It is then easy to acquire a collection of related constructions without a clear account of why they belong together.

The organising question of this tutorial is deliberately simple:

What is a stateful system *between two theories*?

Here “theory” is initially understood in the computational-effects sense: a monad or Lawvere theory specifying available operations and the equations they satisfy. Later we will change the ambient category so that the same word can describe resource-sensitive, certified, concurrent, or otherwise enriched theories.

2 The basic formula

Let T and R be monads on **Set**. A residual stateful runner from T to R , with state set S , is given by a natural family

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

compatible with the units and multiplications of the two monads.

The formula is worth reading before doing anything abstract with it. A computation arrives expressed in the language T . The runner is in some hidden state $s \in S$. The runner interprets the incoming computation, possibly updates its own state, and emits an R -computation. The residual effect R has not necessarily been discharged. It may be handled later, or passed through another runner.

Thus a runner is not merely a map of effect theories. It is a *stateful implementation* of one theory in another.

When $R = \text{Id}$, the formula becomes

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

Now the machine completely executes the T -effect. For algebraic theories this is the familiar world of comodels. Ordinary comodels therefore sit at one boundary of a two-variable construction rather than forming an isolated notion.

A useful comparison. A direct theory morphism $T \rightarrow R$ says how to translate every T -operation into an R -term without hidden state. A runner says how to do the same job while retaining private state across interactions. The tutorial will eventually treat these as two different directions—*tight* and *loose*—of the same architecture.

3 A first familiar machine appears

The abstract formula quickly produces ordinary state machines. Consider the theory generated by

- an input-indexed family of unary operations step_a , for $a \in A$, and
- one O -ary operation get , subject to the reader equations ensuring that observation is repeatable and does not itself change state.

Call the resulting theory $T_{\text{Moore}}(A, O)$.

A $\{T_{\text{Moore}}(A, O), \text{Id}\}$ -runner is exactly a Moore machine:

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

The unary operation step_a updates the state, while get reports $o(s)$ and leaves the state alone.

This example is deliberately modest. The point is not that category theory has discovered a new definition of a Moore machine. The point is that a familiar machine has appeared as a special case of the question

“how can one theory be statefully realised in another?”

Once we keep both theory variables, we can ask new questions without changing the basic definition. What if the residual theory is probability? Nondeterminism? Store? A cost theory? Another machine theory?

4 The Sweedler hom packages all runners at once

The connection with Sweedler theory is stronger than an analogy. In the category of accessible endofunctors of **Set**, McDermott, Rivas and Uustalu show that there is a measuring comonad

$$\mathcal{M}(T, R)$$

whose coalgebras are precisely the residual stateful runners from T to R . Thus

$$\text{Coalg}(\mathcal{M}(T, R)) \simeq \text{stateful runners } T \rightsquigarrow R.$$

Classical Sweedler theory studies universal measurements between algebraic objects. Here the measuring object itself is a comonad, and choosing one of its coalgebras amounts to choosing a concrete stateful implementation. That is the structural fact around which the tutorial is organised.

It suggests a useful change of emphasis. Instead of viewing $\mathcal{M}(T, R)$ merely as an auxiliary construction attached to two monads, we can read it as a universal space of possible implementations of T -behaviour in an R -environment.

5 The proposed architecture: $\mathbf{Run}(\mathcal{D})$

Sweedler theory is naturally enriched: measuring comonoids compose. This leads to the systems-theoretic proposal that, for a suitable duoidal category $\mathcal{D}$, there should be a double-category-like object

$$\mathbf{Run}(\mathcal{D}).$$

Its intended dictionary is:

Role	Algebraic object	Systems reading
Objects	monoids/theories T in $\mathcal{D}$	kinds of interaction or effectful interface
Tight arrows	theory morphisms $T \rightarrow R$	stateless interpretation, compilation, refinement, wiring of operation languages
Loose arrows	coalgebras realising $\mathcal{M}(T, R)$	stateful runners $T \rightsquigarrow R$
2-cells	compatible maps between runners	simulations or transformations respecting boundary theory maps
Loose composition	Sweedler/enriched composition	execute one implementation through another while retaining both hidden states

This separation between tight and loose directions is one of the main conceptual advantages of the proposal. Changing a language of operations is not the same thing as supplying a stateful mechanism that implements it. Yet the two processes should interact coherently. There are two basic forms of this interaction. First, loose arrows compose with loose arrows. If

$$T \rightsquigarrow R \quad \text{and} \quad R \rightsquigarrow P$$

have state spaces S and Q , their composite has, in the elementary Set picture, state $S \times Q$. One first runs the T -computation through S , producing an R -computation, then runs that through Q . The associativity of loose composition is therefore the associativity of layered execution.

Second, tight maps change the theories at the boundaries of a runner. Given theory maps on the source and target, a compatible 2-cell says that one stateful realisation refines, transports, or compares with another while respecting those boundary interpretations. This is where the genuinely double-categorical picture enters: tight theory change and loose stateful execution are not two unrelated constructions, but the vertical and horizontal directions of one calculus. In applications, this lets us alter or wire interfaces while still asking how the implementations behind them are related.

One slogan for the series. Tight maps say how operation languages are reinterpreted. Loose maps say how one operation language is *realised by a stateful mechanism* in another. The proposed double category keeps these two ideas distinct while allowing them to interact.

6 Runners also act on ways of using a system

A runner is not only an operational object. Once a system has been implemented, we still want to know what can be *done* with it: how its remaining effects are answered, observed, evaluated, or turned into outcomes. This is the role of a handler. An algebra for a theory is a way of giving concrete meaning to its operations on some carrier of results. Thus an R -handler describes a context capable of dealing with the residual requests that the runner leaves behind.

Suppose a runner $T \rightsquigarrow R$ has hidden state S , and let C be an R -algebra. A user presents a T -request. The runner translates that request into an R -computation while changing its private state; the R -handler then deals with the residual computation. To use this composite repeatedly, however, one must remember which runner state has been reached. A T -computation is therefore interpreted not merely as an element of C , but as a state-indexed continuation in

$$C^S.$$

This gives the runner-induced functor

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

It is useful to think of this as *delegation*: a context that knows how to handle R can handle T by delegating each T -interaction to the runner and then handling what remains.

In many Set-based examples this functor has a left adjoint. If delegation asks how an R -handler can be pulled back across an implementation, the left adjoint asks the converse systems question: given a way of using the abstract T -system, what concrete R -system results when we run it against the implementation? We call this *induction* or *execution*. In the comodel case $R = \text{Id}$, it is the familiar model–comodel tensor.

This gives two complementary views of the same runner:

operationally: it realises T -requests by R -effects,
 semantically: it transports ways of handling those effects.

The point becomes especially concrete for probability and nondeterminism. The same stochastic runner can be evaluated by expectation, entropy, or another algebra; the same nondeterministic runner can support may- or must-style semantics. The implementation is held fixed while the question asked of it changes.

7 Why probability and nondeterminism are revealing

Once the residual theory is allowed to vary, familiar kinds of systems appear without rebuilding the framework.

For example, let `rand` be a finite-lottery monad. A Moore runner

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

contains stochastic state transitions

$$\delta : A \times S \longrightarrow \mathbf{rand}(S)$$

while the reader equations force observation itself to remain deterministic and state-preserving. Different **rand**-handlers then induce different semantics on the same stochastic implementation: expectation yields Markov/backward operators; an entropy algebra yields immediate entropy plus expected continuation cost.

If we replace **rand** by the nonempty-list monad, the same shape describes nondeterministic transition. Choosing Boolean disjunction, Boolean conjunction, min, or max as residual handlers turns one runner into may, must, angelic, or demonic-style semantics.

The lesson is important for the rest of the series:

runner = implementation; residual handler = way of evaluating that implementation.

This separation is one reason the two-variable Sweedler hom is more expressive than studying comodels one theory at a time.

8 The ambient category is part of the modelling choice

The first examples live in

$$\mathcal{D} = [\mathbf{Set}, \mathbf{Set}]_{\text{acc}}.$$

But this is not meant to be the final universe. A major theme of the later tutorial is that changing $\mathcal{D}$ changes what counts as a theory, a stateful implementation, and a residual effect.

For Sweedler theory we need more than a single monoidal product. A suitable *duoidal* category carries two compatible monoidal structures. In the endofunctor example one product is composition, whose monoids are monads. The second is a symmetric convolution product controlling interaction and measuring. The interchange law says, roughly, that interaction is compatible with substitution of computations.

This machinery is deliberately postponed until the reader has already met concrete runners. Once it is exposed, however, it becomes a checklist for moving to other semantic worlds.

It is helpful to see the endofunctor category as only the first member of a family. Different choices of $\mathcal{D}$ change what the objects of state can carry and what the two modes of composition mean. Among the examples developed or explored in the series are:

- **valued predicates $\mathbf{Pred}_V$:** states carry truth values, safety information, costs, or other ordered certificates, so running a system can simultaneously propagate logical or quantitative information;
- **copresheaves $\mathbb{C}\text{-Set}$ and their endofunctors:** a monoidal category $\mathbb{C}$ grades states by resources, time, or cost; in the thin cost case this leads to potential-based and amortised semantics, while $\mathbb{C} = 2$ recovers a predicate-like picture;
- **presheaf and higher-dimensional settings:** categories such as precubical sets replace point-like state by cells recording compatible concurrent executions, opening the way from ordinary automata to higher-dimensional automata;
- **species and profunctor-like settings:** convolution and substitution can encode the two compositions appropriate to networks, Petri nets, graphical species, and process theories, where resources are combined in parallel while processes are composed by gluing or substitution;

- **smooth or analytic settings:** replacing sets by a category supporting smooth or analytic structure asks whether runners can encompass ODE-based systems and, more generally, systems whose state spaces carry geometry rather than only combinatorics;
- **categorified or homotopical settings:** replacing sets by categories, groupoids, or anima allows states themselves to have symmetries, identifications, and higher paths, rather than treating equality of states as mere equality of elements.

The examples are deliberately heterogeneous. That is the point: duoidality is not a technical decoration on one favourite category, but the reusable piece of structure that tells us when a semantic universe supports both theories and their Sweedler-style interaction. The practical question in each new setting is therefore not “does it look like $[\mathbf{Set}, \mathbf{Set}]_{\text{acc}}$?” but “what are the two monoidal products, what are their monoids, and what systems meaning does the interchange law acquire?”

So $\mathbf{Run}(\mathcal{D})$ is intended not as one category of systems but as an architecture that can itself vary with the semantic universe.

9 A second axis: theories can vary too

Changing $\mathcal{D}$ is not the only source of flexibility. Within a fixed ambient category, theories can be built, combined and reindexed.

The tutorial therefore distinguishes several constructions that are often all informally called “composition”:

- sequential composition of runners;
- composition of tight theory interpretations;
- coproduct of theories, putting operation vocabularies side by side;
- tensor product of theories, forcing operations from two theories to commute;
- pushouts and coequalisers, gluing theories or imposing equations;
- parameter change, for families such as $T_{\text{Moore}}(A, O)$.

These distinctions matter. A wiring diagram may tell us how ports are interconnected; a theory tells us what operations exist and what equations they satisfy; a runner tells us how those operations are implemented through hidden state. The series tries to keep those layers separate long enough to understand how they fit together.

10 What the tutorial is trying to teach

This is not primarily a survey of abstract Sweedler theory. The order of exposition is deliberately backwards from what one might choose in a research paper.

We first ask what the definitions do in tiny cases. We compute identity runners, unary dynamics, reader comodels and Moore machines. We then allow nontrivial residual effects. Only after the examples have created a need for the surrounding structure do we introduce tight maps, theory constructions, duoidality, enrichment, and changes of base.

The intended rhythm is:

small example $\longrightarrow$ pattern $\longrightarrow$ categorical structure $\longrightarrow$ new example.

Accordingly, proofs that are essential for orientation appear in the main text; technical existence arguments whose role is only to justify a working ambient category may be moved to appendices. The tutorial is meant to be usable by a reader interested in systems before that reader has mastered all of enriched category theory.

11 A provisional map of the series

The precise ordering may change as the project develops, but the present trajectory is as follows.

Part	Main question
I	How do runners work in the smallest examples, and how does a Moore machine emerge as a $\{T, \text{Id}\}$ -runner?
II	What changes when the residual theory is nontrivial—probability, nondeterminism, store, or another machine theory? How do residual handlers evaluate the same runner in different ways?
III	What are tight maps? How do theory morphisms, coproducts, tensors, pushouts, parameter changes and wiring constructions sit beside loose runners?
IV	Why does the construction require a duoidal category in the first place? What does the interchange law do, how does it generate the monad–comonad interaction underlying runners, and what must be checked before changing the ambient Sweedler universe?
V	What happens when the first change of ambient category is made to predicates and $\mathcal{V}$ -valued predicates? How do invariants, assumptions and guarantees enter certified residual runners, and how does this compare with categorical assume–guarantee reasoning?
VI	How should resource-sensitive and amortised systems be treated? What is the relation between graded objects in $\mathbb{C}\text{-Set}$ and strong or enriched monads on them, and how does the predicate case reappear as a special grading?
VII	How does higher-dimensional execution geometry enter through precubical sets and higher-dimensional automata, and how can concurrent models such as Petri nets feed into it?
VIII	What changes in species and symmetry-rich presheaf universes? How do $\text{Fam}(\mathbf{Group}\text{-}\mathbf{Set}^{op})$, digraphical species, Petri nets and graphical process theories fit the Sweedler pattern?
IX	How do observability, reachability and behaviour arise from the runner/comodel picture? What becomes of Garner’s final behaviour comodel, categories of behaviours, minimal realisation, and the reachability–observability duality?
X	How should smooth systems governed by ODEs fit the architecture, and what do smooth state spaces, smooth Moore machines and Lyapunov-style reasoning add to the picture?

Part	Main question
XI	How are open runs closed into loops? What do behavioural profunctors, shadows, traces, wheeled props and process valuations contribute to feedback and cyclic behaviour?
XII	What might an internal language for Sweedler system theory look like, and how does it connect the semantic architecture back to syntax, including resource-sensitive and homotopical type-theoretic ideas?

The later parts increasingly test whether structures first seen in elementary Set-based examples survive change of base. Part IV isolates the duoidal mechanism that makes the Set-based theory work; Parts V–VIII then change the ambient world in several different directions; Part IX returns to reachability, observability and behaviour; and Parts X–XII move through smooth dynamics, cyclic or traced behaviour, and internal language. These are treated not as disconnected applications but as stress tests for the same architecture.

12 Established mathematics and proposed architecture

Because the tutorial is attached to an active research programme, it is useful to separate three levels of confidence.

12.1 Established ingredients

The underlying theory of stateful runners/comodels, the Sweedler theory of monads, and the more general theory of measuring comonoids in suitable duoidal categories are established mathematics. In the accessible-endofunctor setting, residual stateful runners are classified by coalgebras of the relevant Sweedler hom comonad.

The tutorial also uses standard constructions around Lawvere theories, monads, Day convolution, enriched categories, and locally presentable categories. When later chapters move to an enriched or resource-sensitive base, some of the surrounding machinery is standard even when the particular systems-theoretic interpretation is new.

12.2 The organising proposal

The proposed object $\mathbf{Run}(\mathcal{D})$ is an architectural reading of these ingredients: theories as objects, theory interpretations as tight arrows, runners as loose arrows, and transformations as cells. Pieces of this structure are already supplied by enrichment and Sweedler composition, but a fully polished functorial double-categorical treatment remains part of the programme.

12.3 Research questions

Some later extensions are deliberately presented as targets rather than finished theorems. Examples include the most general enriched accessible Sweedler construction needed for cost-indexed copresheaves, satisfactory fibred and dependent forms of $\mathbf{Run}$ for varying interfaces, higher or homotopical measuring objects, and the precise scope of the passage from runners to behavioural profunctors and their shadows.

The tutorial will mark these transitions rather than silently treating conjectural architecture as settled fact.

13 A philosophical reading, if one wants one

There is a systems-theoretic interpretation that will occasionally guide our intuition.

A theory T specifies a kind of agency: which requests may be made and which equations those requests obey. A runner $T \rightsquigarrow R$ says how that agency can be embodied using a different layer of capabilities R , together with some hidden organisation S . An R -handler supplies a further semantic or material context in which those residual capabilities are finally interpreted.

Thus one can have chains

$$T \rightsquigarrow R \rightsquigarrow P \rightsquigarrow \dots$$

without declaring any single layer to be “the” physical substrate. Implementation is relative. A Moore interface may be realised by probabilistic dynamics; those probabilistic effects may themselves be realised by a lower-level random generator; a certified or resource-sensitive ambient category may constrain every stage.

Nothing in the mathematics requires this philosophical language. But it helps explain why the two-variable construction matters: a system is not characterised only by what it does in isolation, but also by *what language it receives and what language it leaves behind*.

14 How to read the tutorial

The safest route is simply to read in order. Parts I and II establish the operational intuition that later abstractions depend on. Part III can be read as the companion chapter on stateless theory change. Part IV then steps back and explains the duoidal machinery that has been operating in the background. Part V makes the first genuine change of ambient Sweedler universe, to predicates and $\mathcal{V}$ -valued predicates; Parts VI–VIII continue the change-of-base programme through resources, higher-dimensional concurrency, and symmetry-rich species. Part IX is where reachability, observability and behaviour become central, before the final three parts turn to smooth dynamics, traces and loops, and internal language.

Readers who already know algebraic effects may skim some of the elementary monad examples but should keep an eye on the direction of the handler functor: a runner with state S sends an R -handler with carrier C to a T -handler on C^S , not on $C \times S$. The product appears instead in execution and in composition of hidden states.

Readers coming from applied category theory may find it useful to keep three questions separate throughout:

1. What is the *interface or theory* presented to the environment?
2. What *stateful mechanism* realises that theory?
3. What *semantic handler or residual environment* finally interprets the remaining effects?

Most of the apparent complexity of the subject comes from allowing all three to vary independently.

The shortest preview. A runner is a stateful implementation of one theory in another. Sweedler theory provides a universal object collecting such implementations. The proposed **Run**($\mathcal{D}$) architecture separates stateless theory interpretation from stateful realisation, lets implementations compose, and allows the whole construction to be moved into richer semantic worlds such as predicates, resources, concurrent geometry, or process categories. The tutorial develops that claim one example at a time.

References and orientation

- T. Uustalu, *Stateful runners of effectful computations*, 2015.
- D. McDermott, E. Rivas and T. Uustalu, *Sweedler Theory of Monads*, FoSSaCS 2022.
- I. López Franco and C. Vasilakopoulou, *Duoidal categories, measuring comonoids and enrichment*, 2020.
- R. Garner, *The costructure–cosemantics adjunction for comodels for computational effects*, 2020.
- Later parts of the tutorial add references specific to probability, predicates, amortisation, Petri nets, higher-dimensional automata, and reachability–observability.