Getting to Grips with Sweedler Runners for Systems

A conversational tutorial, Part XII: an internal language for running theories
Commands, capacities, residual effects, and scoped execution

Revised 14 November 2026

Aim. What would a programmer write in a language for running theories? We organise the answer around commands T, capacities D, and residual effects R. A measuring law specifies their interaction; a realisation supplies a machine; a scoped execution construct controls access to its state. Ahman–Bauer's λ_{coop} provides an operational precedent, while the Quantum Systems Language QS illustrates how modal programming notation can arise from indexed categorical structure. We develop a precise Set-based core and distinguish it from the proposed resource-sensitive and dependent extensions.

1 Three roles and two structural sorts

1.1 The interaction comes before its representation

The informal picture is a familiar one: a command makes a demand, an implementation supplies capacity, and meeting the demand may leave further work to do. In our notation these roles are T, D, R. The same theory may occupy either monad position: an R-effect produced by one implementation becomes the command of the next. We therefore distinguish two structural sorts, monads and comonads, rather than declaring command theories and residual theories to be different kinds of object.

Fix an admissible symmetric duoidal category $\mathcal{D}$, with products denoted $\circ$ and $\star$. Our theories are $\circ$-monoids and capacities are $\circ$-comonoids. In the main example, $\mathcal{D} = [\mathbf{Set}, \mathbf{Set}]_{\mathrm{acc}}$, these are monads and comonads; $\circ$ is endofunctor composition and $\star$ is Day convolution for cartesian product. We retain this convention throughout; other references use different product symbols.

A *measuring law* is a compatible map

$$\alpha : T \star D \longrightarrow R.$$

The compatibility uses the monad structures of T, R, the comonad structure of D, and duoidal interchange. In the endofunctor example its components are natural maps

$$\alpha_{A,B} : TA \times DB \longrightarrow R(A \times B).$$

The corresponding prospective judgement is

$$\alpha : D \Vdash T \rightsquigarrow R.$$

This is a judgement about a specified interaction. A capacity D on its own does not determine which commands it implements or how it produces residual effects.

1.2 The universal object and a concrete realisation

Sweedler theory represents these laws by a measuring comonoid [5]:

$$\mathrm{Meas}(D; T, R) \cong \mathbf{Comon}_{\circ}(\mathcal{D})(D, \{T, R\}).$$

Write $\widehat{\alpha} : D \to \{T, R\}$ for the representing map. Where D acts as a comonad on a category of states, a realisation is a coalgebra

$$c : S \longrightarrow DS.$$

Corestriction along $\widehat{\alpha}$ gives a $\{T, R\}$-coalgebra on S, hence a runner in the sense developed earlier. It is useful to keep all four pieces of data distinct:

Data	Role
D	capacity interface
$\widehat{\alpha} : D \to \{T, R\}$	chosen implementation law
$c : S \to DS$	realisation on a state object
$s : S$	current configuration

In an arbitrary duoidal category a comonoid need not itself be a comonad acting on ordinary states. The coalgebra notation above is literal in the endofunctor setting; other settings require an appropriate action or comodule notion. This distinction prevents the abstract capacity object from being silently identified with a Set-valued state space.

Where convolution and the copower exist with the required structures, the same interaction has the other two presentations

$$T \longrightarrow [D, R], \qquad D \triangleright T \longrightarrow R.$$

These are transposes of one measuring law, not three independent implementation choices. Their existence must be checked in the category being used; an accessible-endofunctor construction need not remain inside polynomial endofunctors or polynomial monads.

2 Theories supply effect syntax

2.1 Operations, equations, and translations

An algebraic presentation has operations with parameter sets A_{op} and response sets B_{op}. A term

$$\mathrm{op}(a, b \mapsto m_b)$$

issues a request, receives a response, and continues. Equations identify computations, and the resulting monad T records substitution. We write $\Gamma \vdash_T m : A$ for a computation returning an A, and $\Gamma \vdash v : A$ for a value. These are schematic judgements relative to an ambient value language.

A tight map $f : T \to R$ translates operations into residual terms, respecting equations and substitution. It acts by

$$\frac{\Gamma \vdash_T m : A}{\Gamma \vdash_R f_* m : A}.$$

For $T = R = Ny$, a fixed translation can replace every tick by k ticks. A stateful implementation can choose that number from its current configuration. The latter needs the interaction and realisation data of the previous section.

2.2 Operations supplied by ambient structure

Not every effect language must start from a separately chosen signature. An adjunction already supplies a monad, and indexed categories often supply several related ones. QS will provide an important example in Section 8. Algebraically presented theories and structurally induced modalities are complementary sources of computation judgements. Neither source automatically supplies a measuring law with an arbitrary capacity.

3 Store, Yoneda, and state-passing execution

3.1 A precise representation calculation

The store comonad for a set S is

$$\mathrm{Store}_S\, B = S \times B^S, \qquad \varepsilon_B(s,g) = g(s), \qquad \delta_B(s,g) = (s, u \mapsto (u,g)).$$

An element combines a current position s with an observation map g for possible positions. This is a comonad on result types, not the state monad. The relationship between interaction laws and stateful runners is studied by Katsumata–Rivas–Uustalu [4].

Proposition 3.1 (Store representation)**.** *For endofunctors T, R on* **Set***, natural families*

$$\alpha_{A,B} : TA \times \mathrm{Store}_S\, B \longrightarrow R(A \times B)$$

correspond bijectively to natural families

$$\theta_A : TA \times S \longrightarrow R(A \times S).$$

Under this correspondence, measuring compatibility is equivalent to the state-passing unit and multiplication laws displayed below.

Proof. For fixed A, curry in $TA \times S$ and apply Yoneda to $\mathbf{Set}(S, B)$. Explicitly the two directions are

$$\theta_A(t,s) = \alpha_{A,S}(t,(s,\mathrm{id}_S)), \qquad \alpha_{A,B}(t,(s,g)) = R(\mathrm{id}_A \times g)\theta_A(t,s).$$

Naturality in B makes them inverse, and naturality in A is preserved. The measuring unit uses the store counit and becomes

$$\theta_A(\eta^T_A(a), s) = \eta^R_{A\times S}(a,s). \tag{1}$$

For multiplication, the componentwise measuring equation is

$$\alpha_{A,B}(\mu^T_A(t), d) = \mu^R_{A\times B} R(\alpha_{A,B})\big(\alpha_{TA,DB}(t, \delta^D_B(d))\big).$$

For store, $\delta(s,g) = (s, u \mapsto (u,g))$. The right-hand side therefore reduces, by the reconstruction formula and naturality of μ^R, to

$$R(\mathrm{id}_A \times g)\big(\mu^R_{A\times S} R(\theta_A)\theta_{TA}(t,s)\big).$$

The left-hand side is $R(\mathrm{id}_A \times g)\theta_A(\mu^T_A(t), s)$. Taking $B = S$ and $g = \mathrm{id}_S$ gives

$$\theta_A \circ (\mu^T_A \times \mathrm{id}_S) = \mu^R_{A\times S} \circ R(\theta_A) \circ \theta_{TA}. \tag{2}$$

Conversely, substitution of the formula for α into its unit and multiplication diagrams reduces them to these equations, using naturality and the displayed store structure. Since the Yoneda correspondence is injective, their evaluations determine the diagrams. □

Equivalently, currying θ gives a monad map

$$T \longrightarrow \mathrm{St}_S^R, \qquad \mathrm{St}_S^R\, A = \big(R(A \times S)\big)^S,$$

where the state-transformer multiplication threads state through residual Kleisli substitution. Thus the compatibility claim is stronger than a bijection between unstructured natural transformations.

3.2 Where the finaliser enters

The set S has its canonical store coalgebra $s \mapsto (s, \mathrm{id}_S)$. The corresponding execution map is θ. It exposes final state to the *implementation boundary*; it need not expose that state to client code.

Let $q : A \times S \to B$ be a pure finaliser. Scoped execution has denotation $R(q)\theta_A$. To recover it directly from the store slot, put

$$\widehat{q} : S \longrightarrow B^A, \qquad \widehat{q}(s)(a) = q(a, s).$$

Then

$$R(\mathrm{ev})\alpha_{A,B^A}(t, (s, \widehat{q})) = R(q)\theta_A(t, s), \qquad \mathrm{ev}(a, h) = h(a).$$

For an effectful finaliser $q : A \times S \to RB$, the same calculation gives

$$\mu_B^R R(\mathrm{ev})\alpha_{A,(RB)^A}(t, (s, \widehat{q})) = q^{\dagger_R}\theta_A(t, s), \tag{3}$$

where $q^{\dagger_R} = \mu_B^R R(q)$. The function component of store therefore accommodates finalisation by continuation. It does not force a particular cleanup policy, privacy discipline, or exactly-once theorem: those require further choices about the language and residual effects.

4 Scoped execution and λ_{coop}

4.1 Private state belongs to the implementation scope

Ahman–Bauer distinguish values, user computations, and kernel computations, and provide the construct

$$\mathsf{using}\ V\ @\ W\ \mathsf{run}\ M\ \mathsf{finally}\ F.$$

Co-operation clauses run in kernel mode, with private state; finalisation deals with the possible outcomes [3]. This is the closest operational precedent for our initial calculus.

For the fragment with ordinary returns, write $V : T \Rightarrow (R; S)$ for a declared store-based implementation. A candidate scoped rule is

$$\frac{\Gamma \vdash V : T \Rightarrow (R; S) \qquad \Gamma \vdash W : S \qquad \Gamma \vdash_T M : A \qquad \Gamma, x : A, c : S \vdash_R N : B}{\Gamma \vdash_R \mathsf{using}\ V@W\ \mathsf{run}\ M\ \mathsf{finally}\{x@c \mapsto N\} : B}.$$

Here B is formed outside the scope of c. Client code M receives no binding for the private state; the finaliser receives both the return value and the final state. Its semantics is exactly (3). The exposed map $TA \times S \to R(A \times S)$ remains useful as an intermediate semantic operation, but is not the public result type of this construct.

This simple rule does not establish ownership for arbitrary values already present in Γ, nor forbid a finaliser from deliberately returning state when the types allow it. Stronger representation hiding requires abstract state types or a package such as

$$\exists S.\big(\mathsf{Impl}(T, R; S) \times S\big),$$

with a suitable elimination rule. Linear ownership requires an additional discipline. State scoping, type abstraction, and ownership are distinct design decisions.

4.2 Co-operation clauses and computation rules

For a free source signature, an implementation supplies clauses

$$\kappa_{\mathrm{op}} : A_{\mathrm{op}} \times S \longrightarrow R(B_{\mathrm{op}} \times S).$$

They extend recursively by

$$\begin{aligned}
&\theta_A(\mathsf{return}\ a, s) = \mathsf{return}_R(a, s),\\
&\theta_A(\mathrm{op}(a, k), s) = \kappa_{\mathrm{op}}(a, s) \gg\!=_R \big((b, s') \mapsto \theta_A(k(b), s')\big).
\end{aligned}$$

The implementation clause receives the request and state; it does not receive the client continuation as an arbitrary higher-order argument. To descend to a theory with equations, its induced interpretation must respect those equations. This is a proof obligation, not a consequence of merely listing clauses.

Two useful scoped equations follow immediately:

$$\begin{aligned}
&\mathsf{runFinally}_V(s, \mathsf{return}\ a, q) = q(a, s),\\
&\mathsf{runFinally}_V(s, \mathrm{op}(a, k), q) = \kappa_{\mathrm{op}}(a, s) \gg\!=_R \big((b, s') \mapsto \mathsf{runFinally}_V(s', k(b), q)\big).
\end{aligned}$$

These equations provide a precise small core before first-class capacities, dependency, or multiple resource contexts are added.

4.3 Exceptions, signals, and the extent of the guarantee

In λ_{coop}, exceptions preserve state, whereas fatal signals discard it. With outer operation signature Σ, exception set E, signal set K, and state S, its kernel monad has shape

$$S \longrightarrow \mathrm{Tree}_{\Sigma}\big(((A + E) \times S) + K\big).$$

The finaliser has return and exception clauses with state, and signal clauses without state. Theorem 7 of the arXiv v2 paper factors execution through finalisation; its guarantee is qualified by external kill signals [3]. It is not an unconditional termination theorem.

Our basic store representation always returns state on successful residual results. To model state-discarding signals faithfully, one must change the outcome/state arrangement or use a richer interface. Simply adding an exception operation while continuing to return S in every branch misses this distinction. Reading a signal as termination on the capacity side is a possible future interpretation, not an established classification of signals by comonads.

5 Composition: nested scopes and residual work

5.1 The capacity-level composite

If

$$\widehat{\alpha} : D \to \{T, R\}, \qquad \widehat{\beta} : E \to \{R, P\},$$

enriched composition supplies

$$E \star D \longrightarrow \{R, P\} \star \{T, R\} \longrightarrow \{T, P\}.$$

Equivalently, up to the symmetric reassociation convention, one pairs T with D, obtains R, then pairs R with E. This states composition for capacities without presupposing a particular state representation.

For realised store runners $Y : T \rightsquigarrow R$ on S and $Z : R \rightsquigarrow P$ on U, the composite state-passing map is

$$TA \times S \times U \xrightarrow{\theta^Y_A \times \mathrm{id}_U} R(A \times S) \times U \xrightarrow{\theta^Z_{A \times S}} P((A \times S) \times U).$$

Reassociation identifies its state with $S \times U$. This construction gives a map from the two capacity descriptions to the store capacity of combined state; it does not assert $\mathrm{Store}_U \star \mathrm{Store}_S \cong \mathrm{Store}_{S \times U}$.

5.2 A law with finalisers

Let $q : A \times S \to RB$ be the inner finaliser and $r : B \times U \to PC$ the outer one. Nested execution means

$$\mathsf{runFinally}_Z(u, \mathsf{runFinally}_Y(s, m, q), r).$$

It agrees with composite execution followed by

$$h(a, s', u') = \mathsf{runFinally}_Z(u', q(a, s'), r).$$

Indeed, the equality is (2) for Z, applied to residual Kleisli substitution by q. The outer runner must implement residual effects of the inner finaliser as well as those of the client execution. This is why an unqualified formula $\mathsf{run}_Z \circ \mathsf{run}_Y$ suppresses essential scoping information. The displayed law is for our return-only fragment; extending it to signals requires the richer outcome semantics.

5.3 Ticks as a running example

A $\{Ny, Ny\}$-runner is described by

$$f : S \to S, \qquad k : S \to \mathbb{N}.$$

One source tick updates s to $f(s)$ and emits $k(s)$ residual ticks. Put

$$K_n(s) = \sum_{i=0}^{n-1} k(f^i(s)).$$

Executing n ticks sends (a, s) to $(a, f^n(s))$ with residual tick count $K_n(s)$. If the next runner on U has maps g, ℓ, one source tick in the composite has state update and residual count

$$(s, u) \longmapsto (f(s), g^{k(s)}(u)), \qquad \sum_{j=0}^{k(s)-1} \ell(g^j(u)).$$

The empty sum is zero. Finalisers can consume or summarise the resulting private states without making them part of the client interface. This example distinguishes state-dependent implementation from a fixed monad translation and makes residual work visible.

6 Duoidal structure as compatibility of execution

In the endofunctor model, $\circ$ records nesting and substitution, while Day convolution packages the two-input interaction $TA \times DB \to R(A \times B)$. Its interchange map has the form

$$(F \circ G) \star (H \circ K) \longrightarrow (F \star H) \circ (G \star K).$$

For a measuring, the multiplication equation is precisely

$$\begin{aligned} \alpha \circ (\mu^T \star \mathrm{id}_D) = \mu^R \circ (\alpha \circ \alpha) \circ \zeta \\ \circ (\mathrm{id}_{T \circ T} \star \delta^D). \end{aligned}$$

Here $\alpha \circ \alpha$ denotes horizontal composition of natural transformations; all reassociations are understood. The source on both sides is $(T \circ T) \star D$. The comultiplication supplies capacity for successive stages; interchange rearranges the two interactions; multiplication collects the residual computations. This is the precise content behind a proposed reading as a cut/substitution compatibility law.

It is premature to identify this interchange map with a general commuting conversion in a finished calculus, or to infer two term-context formers just because $\mathcal{D}$ has two products. The two modes of λ_{coop} are likewise not simply the two duoidal products. Multiple contexts become justified when the ambient value/resource semantics requires them. Bunched syntax remains a promising technology for that later task.

7 Handlers and transport beyond scoped execution

An algebraic handler interprets a theory in a chosen carrier; a runner implements it while leaving residual effects. General handlers can also transform one effect language into another [1]. The distinction is therefore their permitted structure and continuation use, rather than the mere presence of residual effects. λ_{coop} chooses a controlled runner fragment to obtain its resource discipline.

For a store runner θ and an R-algebra $b : RB \to B$, delegation gives a T-algebra on B^S. Its structure sends $t \in T(B^S)$ to the function

$$s \longmapsto b(R(\mathrm{ev})(\theta_{B^S}(t, s))), \qquad \mathrm{ev}(v, s') = v(s').$$

Thus $\Theta_Y : \mathbf{Alg}(R) \to \mathbf{Alg}(T)$ is more than the single scoped execution construct. In the settings of the earlier parts it has a left adjoint L_Y, with

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

This supports the investigation of runner-indexed modalities. An adjunction between algebra categories alone does not yet supply type-former rules in every context. An internal account must specify context interpretation, substitution, and the stability of these constructions. This layer goes beyond the execution constructs borrowed from λ_{coop}.

8 What QS contributes

8.1 A language derived from indexed structure

Sati–Schreiber's *Quantum Monadology* derives a system of modal effects from parameterised linear categories, and sketches QS notation expressible in dependent linear type theory [11]. Its design starts from base-change operations

$$f_! \dashv f^* \dashv f_*$$

and linear tensor/internal hom. Units, counits, and effect binding acquire programming notation. Its intended formal embedding should be distinguished from a completed implementation of a certified QS language.

The methodological lesson is concrete: an internal runner language can be a small domain-specific layer over an ambient language that already supplies dependency and resources. We need not reconstruct those structures as primitive runner syntax. There is a related classical precedent: a polynomial diagram $I \xleftarrow{s} E \xrightarrow{p} B \xrightarrow{t} J$ in a locally cartesian closed category denotes

$$\Sigma_t \Pi_p s^* : \mathcal{E}/I \longrightarrow \mathcal{E}/J$$

[6]. This construction uses base change and its adjoints, but requires a chosen diagram. It neither canonically selects an effect theory nor automatically supplies its measuring laws. Linear six-operation formalisms and classical polynomial semantics also have different hypotheses.

8.2 Adjoint monads and comonads: a special evaluation

For clarity, let $L : \mathcal{A} \to \mathcal{B}$, $F : \mathcal{B} \to \mathcal{A}$, and $G : \mathcal{A} \to \mathcal{B}$, with $L \dashv F \dashv G$. Then

$$T_0 = FL \quad \text{is a monad on } \mathcal{A}, \qquad D_0 = FG \quad \text{is a comonad on } \mathcal{A}.$$

There is an adjunction $T_0 \dashv D_0$:

$$\mathcal{A}(FLX, Y) \cong \mathcal{B}(LX, GY) \cong \mathcal{A}(X, FGY).$$

The monad and comonad structures are mates. More generally, if a monad T_0 has a right adjoint D_0, the mate structure makes D_0 a comonad and gives an isomorphism

$$\mathbf{Alg}(T_0) \cong \mathbf{Coalg}(D_0)$$

over the underlying category [7, 8]. For unit $u : \mathrm{Id} \to D_0T_0$ and counit $e : T_0D_0 \to \mathrm{Id}$, the corresponding structures are

$$(a : T_0X \to X) \longmapsto D_0(a)u_X, \qquad (c : X \to D_0X) \longmapsto e_X T_0(c).$$

The counit is a *nested* evaluation $T_0D_0 \to \mathrm{Id}$, compatible with the mate structures. Its tensor is composition. Our principal measuring instead has shape $T \star D \to R$, with Day convolution in the endofunctor example. Passing from nested evaluation to a Day measuring requires additional comparison data and coherence; the adjunction alone does not provide that passage.

8.3 Finite linear parameters and ambidexterity

Take $p : W \to 1$ with W finite and consider vector spaces indexed by W. Then

$$p^*V = (V)_{w\in W}, \qquad p_!(V_w) = \bigoplus_w V_w, \qquad p_*(V_w) = \prod_w V_w.$$

The canonical finite biproduct comparison identifies $p_!$ and p_*. On **Vect**, the monad p_*p^* and comonad $p_!p^*$ consequently share an underlying functor $V \mapsto \bigoplus_W V$, with Frobenius structure. Explicitly, for $C_W = \bigoplus_W k$, its basis satisfies

$$e_w e_v = \delta_{wv} e_w, \qquad 1 = \sum_w e_w, \qquad \Delta(e_w) = e_w \otimes e_w, \qquad \epsilon(e_w) = 1.$$

These give the monad/comonad on $C_W \otimes (-)$. QS exploits related ambidexterity for finite classical parameters [11]. Stability alone does not identify arbitrary indexed sums and products.

This special coincidence does not abolish the general Sweedler asymmetry. An equivalence between models and comodels of an adjoint pair already exists without ambidexterity; finite ambidexterity additionally identifies particular underlying functors. Neither identifies arbitrary command and residual theories, makes private state public, or supplies every triadic measuring. Appendix C can be read as a specialised linear setting in which some of these extra identifications become available.

8.4 Observation and dynamic lifting

QS treats measurement outcomes as classical indices for families of linear types, and develops dynamic lifting and deferred-measurement principles from its modal structure [11]. For runners, this suggests a useful dependent goal: a returned observation may determine the type of subsequent state or the commands available next. Schematically the outcome then has type

$$\sum_{b:B} S_b$$

rather than a product with a fixed state type. A successful connection can create a session interface; failure leaves a disconnected interface. Our two-world examples make this distinction operationally visible.

Returning an ordinary value to client code is not yet a complete dynamic-lifting rule: one must specify its binding scope and how subsequent types and effect interfaces depend on it. Nor does QS prove a general deferred-observation or deferred-finalisation theorem for runners. A Moore readout can affect the next branch, and a finaliser may have ordered residual effects. Deferral requires explicit hypotheses about observation, commutation, and what equivalence is being asserted.

9 Resources and dependency as extensions

9.1 Ownership, bunched contexts, and Riley

The small calculus above has private state by scope. In λ_{coop}, controlled state passing supplies resource discipline without a linear variable context. If state itself contains affine ownership, graded potential, or dependent linear data, additional ambient structure becomes relevant.

Riley's bunched dependent type theory provides a developed approach to combining dependency with resource-sensitive type formers [10]. His unpointed discussion of parameterised

Day-convolution models is particularly suggestive. It offers possible machinery for contexts such as $\Gamma; \Delta$, whose structural rules differ, and modalities controlling dependency across those regions. These contexts should model the actual value/resource semantics, rather than being inferred directly from the number of duoidal products. Locks and base-change modalities are different available mechanisms; neither is a universal replacement for the other.

9.2 Stored potential and finalisation policies

Let $\mathcal{V} = C$-**Set** carry Day convolution. A resource object $y(p)$ defines

$$P_p A = y(p) \otimes A, \qquad M_p A = [y(p), A], \qquad P_p \dashv M_p.$$

The λ_{amor} calculus uses modal types for stored potential together with affine structure and indexed costs [9]. In Part VI, queue state had the schematic form

$$S = L(P_2\tau) \otimes L(\tau).$$

Potential deposited with an element is retained until a later operation needs it. A resource-sensitive implementation should track this movement in its kernel judgement, with suitable ownership at initialisation and finalisation. Client code need not receive the hidden queue representation.

A graded finaliser might be required to account for remaining potential or establish a zero-debt invariant. Such a requirement must be included in the resource semantics or typing rules: choosing a comonad D does not automatically impose it. Different capacities may support different observation and disposal interfaces, but their connection to finalisation is a research question beyond the store calculation.

9.3 Dependent transport is a further coherence problem

For a dependent language, context substitution should transport theories, capacities, realisations, and measuring laws. We seek judgements such as

$$\Gamma \vdash T : \mathsf{Theory}, \quad \Gamma \vdash D : \mathsf{Capacity}, \quad \Gamma \vdash \alpha : D \Vdash T \rightsquigarrow R.$$

The same Theory sort serves T and R. In slice or indexed semantics, substitution may yield comparison maps between transported measuring objects rather than isomorphisms. The internal grammar must say which substitutions are supported and which coherence conditions hold. The ordinary runner fragment does not settle this issue; the two-world and GAT studies supply tests for it.

10 Changing the ambient world

The triadic architecture can persist while the value and resource language changes. Some examples are:

State/resource world	Structure to expose when available
Set	values, state, operation responses, scoped execution
$\mathbf{Pred}_{\mathcal{V}}$	invariants and quantitative certificates
C-**Set**	potential, grades, resource-sensitive finalisation
Precubical sets	independent transitions and higher-dimensional execution data
Smooth spaces	smooth parameters and tangent-dependent evolution
Indexed linear categories	linear resources, classical indices, modal transport

This table concerns state/resource worlds. Forming a suitable endofunctor category and proving the needed duoidal and representability properties are separate steps. A general ambient category does not automatically provide an algebraic-effects syntax, a dependent calculus, or a behaviour functor.

In particular, a cartesian classical base can index non-cartesian linear fibres. A total category of parameterised spectra may support topos-like semantics while its fibrewise resource tensor remains linear. Classical product, linear tensor, and the products on endofunctors occupy different levels. This is why QS can inform our non-cartesian ambitions without identifying its entire ambient framework with our general $\mathcal{D}$.

11 Execution, hidden representation, and behaviour

An execution calculus constructs and runs implementations. A behaviour logic studies what their observable histories satisfy. In the settings where Part IX supplies behaviour categories and transport to cofunctors or profunctors, that construction gives a second semantic layer; Part XI's shadows then support cyclic behaviour under their own hypotheses. These constructions should not be asserted uniformly for every duoidal ambient category.

There are also two different ways of suppressing state. An existential package hides a carrier from clients while retaining an implementation internally. When a final D-coalgebra exists, the unique map from (S, c) into it records behaviour and may identify observationally indistinguishable states. Existential hiding and passage to final behaviour are not the same construction. Neither the existence of a final coalgebra nor a full-abstraction theorem follows just from the syntax of private state.

A state predicate $P : S \to \mathcal{V}$ and a temporal predicate on trajectories answer different questions. Correct execution of a runner does not by itself establish liveness, recurrence, or a trace valuation. The intended architecture therefore keeps a relationship to prove:

$$\text{scoped execution} \longrightarrow \text{behaviour semantics} \longrightarrow \text{specification and trace reasoning.}$$

The triad helps describe which interface is observed at each step, while behaviour semantics determines what counts as an observable equivalence.

12 A staged programme for Part XII

12.1 The established core

The store representation gives a precise correspondence among measuring laws, coherent state-passing maps, and monad maps into a residual state transformer. For a free signature, co-operation clauses generate those maps; equations impose additional validity conditions. Effectful finalisation is residual Kleisli substitution. Nested execution in the return-only fragment follows the composition law proved above. These are concrete semantic commitments on which to base a calculus.

12.2 The next construction

Specify a small simply typed language of values, theory-indexed computations, implementation declarations, and scoped execution. Its first soundness statement should interpret the return and operation equations, equation-respecting implementation clauses, and composition with nested

finalisers. The tick example provides a complete test case. Exceptions and signals should enter only after the outcome/state semantics is fixed.

12.3 Questions requiring additional work

First, decide whether capacities and measuring witnesses are first-class terms, static declarations, or both. A comonad does not by itself supply a copattern grammar or a cheap cofree construction. Second, relate the store fragment precisely to λ_{coop}, preserving its continuation discipline and the scope of its finalisation theorem. Third, specify ownership and dependent resource substitution, using Riley and amortised calculi where their hypotheses apply.

Fourth, test QS-inspired base-change modalities for actual Day measurings; keep nested adjoint evaluation distinct until the comparison is constructed. Fifth, internalise handler transport only after its behaviour under contexts and substitution is established. Finally, connect execution equivalence with behaviour semantics, rather than equating representation hiding with behavioural quotienting. Any deferred-observation principle should be stated as a separate theorem with explicit assumptions.

> The organising judgement is a specified interaction $D \Vdash T \rightsquigarrow R$. Its realisations execute commands; scoped finalisers control how implementation state crosses the boundary. λ_{coop} guides execution, QS guides modal language design, and Sweedler theory supplies universal representation and composition.

This gives the final part a usable starting point while leaving the dependent, resource-sensitive, and behavioural extensions visible as distinct mathematical tasks.

References

[1] G. D. Plotkin and M. Pretnar. Handling algebraic effects. *Logical Methods in Computer Science*, 9(4), 2013.

[2] T. Uustalu. Stateful runners of effectful computations. *Electronic Notes in Theoretical Computer Science*, 319:403–421, 2015.

[3] D. Ahman and A. Bauer. Runners in action. In *ESOP 2020*, LNCS 12075, pp. 29–55, 2020. doi:10.1007/978-3-030-44914-8_2. Expanded version: arXiv:1910.11629v2; finalisation is Theorem 7 in that version.

[4] S.-y. Katsumata, E. Rivas and T. Uustalu. Interaction laws of monads and comonads. In *LICS 2020*. `https://cs.ioc.ee/~tarmo/papers/katsumata-rivas-uustalu-lics20.pdf`.

[5] I. López Franco and C. Vasilakopoulou. Duoidal categories, measuring comonoids and enrichment. arXiv:2005.01340.

[6] N. Gambino and J. Kock. Polynomial functors and polynomial monads. *Mathematical Proceedings of the Cambridge Philosophical Society*, 154:153–192, 2013. doi:10.1017/S0305004112000394.

[7] S. Eilenberg and J. C. Moore. Adjoint functors and triples. *Illinois Journal of Mathematics*, 9:381–398, 1965.

[8] F. Marmolejo and R. J. Wood. Monads as extension systems—no iteration is necessary. *Theory and Applications of Categories*, 24(4):84–113, 2010. See the introduction for adjoint monads/comonads and their algebra/coalgebra correspondence.

[9] V. Rajani, M. Gaboardi, D. Garg and J. Hoffmann. A unifying type-theory for higher-order (amortized) cost analysis. *Proceedings of the ACM on Programming Languages*, 5 (POPL), 2021. doi:10.1145/3434308.

[10] M. Riley. *A Bunched Homotopy Type Theory for Synthetic Stable Homotopy Theory*. PhD thesis, Wesleyan University, 2022. doi:10.14418/wes01.3.139.

[11] H. Sati and U. Schreiber. The quantum monadology. *Quantum Studies: Mathematics and Foundations*, 12, article 25, 2025. doi:10.1007/s40509-025-00368-5. Preprint: arXiv:2310.15735; QS pseudocode is developed in Section 3 of the May 2025 manuscript.