

Getting to Grips with Sweedler Runners for Systems

A conversational tutorial, Part XII: an internal language for running theories
Effects, implementations, bunched resources, modalities, and the prospect of a Sweedler type
theory

September 2026

Aim. The previous parts have mostly developed a categorical semantics. We have theories, tight interpretations, loose runners, handlers, resource-sensitive variants, behaviour categories, and ways of closing open behaviour into traces. This final chapter turns the question around. If a runner is a stateful implementation of one theory in another, what would a programmer or logician write? The aim is not to announce a finished “Sweedler type theory”. It is to identify a core whose shape is already forced by the categorical semantics, to distinguish this from the choices that remain genuinely syntactic, and to ask how changing the ambient duoidal category should change the resulting language.

1 From categorical semantics to an internal language

A category-theoretic semantics becomes especially useful when it has a corresponding syntax. Cartesian closed categories have simply typed lambda calculi; locally cartesian closed categories support dependent type theory; symmetric monoidal closed categories support linear calculi; monads support effectful computation; fibrations, gradings and modalities support refined forms of dependency. The architecture developed in this tutorial contains several of these structures at once, but it also contains something more specific.

Let $\mathcal{D}$ be an admissible symmetric duoidal category. Its monoids are our theories, and Sweedler theory supplies universal measuring comonoids

$$\{T, R\}.$$

The importance of these objects is not merely that they classify runners one at a time. The category of theories is enriched in comonoids: $\{T, R\}$ is the enriched hom-object from T to R , and there is enriched composition

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

This is the basic structural fact from which the internal-language question should begin [4]. A concrete runner with state S is obtained by giving S the appropriate coalgebra structure for this universal implementation object. In the familiar endofunctor setting it has the shape

$$\theta_X : TX \otimes S \longrightarrow R(X \otimes S).$$

Thus runners are not an unrelated second notion of arrow placed beside theory maps. They are state-bearing realisations of the enriched homs between theories.

This suggests a more precise question than simply asking for a syntax of stateful computation.

What is an internal language for theories whose hom-objects are spaces of implementations?

Some parts of the answer are already determined semantically. Theories must supply effect syntax. Tight maps must act as stateless translations. Coalgebras for $\{T, R\}$ must act as stateful implementations. Sweedler composition must become composition of implementations. Handlers must interpret the residual theory. Other questions remain open: how runner state is bound and hidden, whether it is linear, affine or graded, how dependencies cross resource boundaries, and how the two monoidal structures of the ambient duoidal category should appear in the grammar.

A useful distinction for what follows is therefore

categorical semantics	:	what structures exist and compose,
internal language	:	how those structures are expressed and reasoned about syntactically.

We shall try to keep the two in step. Wherever the semantics already forces a law, the syntax should reflect it. Where several syntactic realisations remain possible, we shall say so.

2 Theories already have syntax

The least speculative part of the story is the theory itself. Suppose T is generated by algebraic operations

$$\text{op} : A_{\text{op}} \longrightarrow B_{\text{op}}.$$

A T -computation can issue an operation, receive a response and continue. Equations between terms specify when two such computations are identified. This is the familiar algebraic-effects point of view [1].

For the Moore theory encountered earlier, for example, the commands were generated from stepping operations together with a read operation, subject to the reader equations on repeated observation. A term of the theory is already a small program: advance the machine, inspect its output, branch on that output, and continue. An algebra

$$h : TC \longrightarrow C$$

then supplies an interpretation of the commands. Plotkin–Pretnar handlers make this operational by giving clauses for the primitive operations and extending those clauses to arbitrary computations.

Thus our prospective language begins with something we already understand:

monoids/theories provide effect syntax.

A tight map

$$f : T \longrightarrow R$$

is then a stateless translation of effect syntax. Each primitive T -operation is replaced by an R -term, and the equations of T must become valid equations in R . For the free theory U on one unary operation u , for instance, the map

$$q_k : U \longrightarrow U, \quad u \longmapsto u^k$$

replaces one abstract step by k concrete steps. If

$$T \xrightarrow{f} R \xrightarrow{g} P,$$

ordinary composition gives the composite translation. Tight maps therefore look syntactically like compilation, macro expansion, or change of instruction set without hidden implementation state.

This already separates two notions that will remain distinct throughout the chapter:

tight maps translate syntax; loose runners implement syntax.

3 Sweedler homs as universal implementation objects

Now let T and R be two theories. The measuring comonoid

$$\{T, R\}$$

is universal for coherent ways of implementing T -behaviour using R -behaviour. A concrete state object S becomes a runner when it is equipped with the corresponding coalgebra structure. It is therefore tempting to call $\{T, R\}$ the “space of implementations of T in R ”, provided we remember that this is an enriched hom-object rather than in general an ordinary type of values.

The important point is that the universal property already tells us what sort of connective a future syntax must somehow expose. One possibility would be a judgement

$$S \Vdash T \Rightarrow R$$

meaning that the resource object S realises an implementation of T in R . Another would be an implementation former

$$[T \Rightarrow R]$$

whose semantic interpretation is derived from $\{T, R\}$. We should not decide the notation too early. What is not optional is the underlying universal property: implementation data from T to R are represented by the Sweedler hom.

Given a particular runner $Y : T \rightsquigarrow R$ with state S_Y , the familiar semantic operation

$$\theta_X : TX \otimes S_Y \longrightarrow R(X \otimes S_Y)$$

has an immediate computational reading. A T -computation producing an X is supplied with private implementation state; the runner turns it into an R -computation producing both the result and an updated state. Schematically one would like a term former of the shape

$$\frac{\Gamma \vdash_T m : A \quad \Gamma \vdash s : S_Y}{\Gamma \vdash_R \text{run}_Y(s, m) : A \otimes S_Y}.$$

This rule suppresses important issues about resource ownership, context splitting and the meaning of $\otimes$, but its operational content is already clear:

a runner executes one effect language by producing residual effects in another.

The elementary unary example makes the distinction from a tight map vivid. Let $T = R = Ny$, the free theory on one unary operation. A stateful runner may be described by functions

$$f : S \longrightarrow S, \quad k : S \longrightarrow \mathbb{N},$$

so that, in state s , one source tick becomes $k(s)$ residual ticks and the private state changes to $f(s)$. The translation chosen for the next source operation can therefore depend on the implementation history. No fixed monad map $T \rightarrow R$ can express this in general.

4 Composition is substitution of implementations

Because $\{T, R\}$ is an enriched hom rather than merely a classifier, runners compose. Suppose

$$Y : T \rightsquigarrow R, \quad Z : R \rightsquigarrow P$$

have state objects S_Y and S_Z . Their composite has combined state

$$S_Y \star S_Z$$

or, in the ordinary **Set**-based case, the corresponding product state. Operationally, the T -computation is first implemented through Y , producing residual R -effects, and those effects are immediately implemented through Z . Both pieces of private state persist.

The syntax should therefore satisfy a law of the schematic form

$$\text{run}_{Z \odot Y}((s, u), m) \simeq \text{run}_Z(u, \text{run}_Y(s, m)),$$

with the appropriate reassociation of returned state. This law is not invented because it seems reasonable operationally. It is the syntactic shadow of Sweedler-enriched composition

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

Associativity and identities for nested implementation should similarly be inherited from the enriched category of theories.

This is already enough to reveal one possible organising idea for the calculus. Ordinary substitution composes terms. Kleisli composition composes effectful computations. Sweedler composition composes implementations. A language for running theories should make all three levels visible without conflating them.

5 Runners are not handlers

The difference between a runner and a handler is particularly important. A handler interprets effects in a semantic carrier. A runner may instead leave residual effects to be interpreted by another runner or by a later handler. Thus we have three levels:

effect syntax	$\longrightarrow$	implementation	$\longrightarrow$	interpretation.
---------------	-------------------	----------------	-------------------	-----------------

A T -term says what abstract command is requested. A runner $Y : T \rightsquigarrow R$ says how that command is realised using the residual theory R . An R -handler finally says what the residual command means.

This separation is familiar from programming-language runners. Uustalu and Ahman–Bauer treat runners as resource-managing implementation layers which answer algebraic operations using co-operations and private state, potentially leaving further external effects to another layer [2, 3]. In our setting the Sweedler hom gives this idea a uniform categorical home.

There is also a stronger semantic structure which any serious internal language should eventually expose. For a runner

$$Y : T \rightsquigarrow R$$

we have repeatedly encountered an adjunction

$$L_Y \dashv \Theta_Y.$$

Here Θ_Y sends an R -handler to the induced T -handler obtained by reasoning through the implementation Y . In the simplest **Set**-based cases its carrier involves a function object such as

$$C^{S_Y}.$$

The left adjoint L_Y moves in the opposite direction. The adjunction

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

therefore says that mapping out of the lowered R -object is equivalent to reasoning about the original T -object through the runner-induced semantics.

Adjunctions of this kind are familiar fingerprints of modal type formers. This suggests that a mature runner calculus should contain more than a primitive run. A runner may index an adjoint pair of modalities expressing two directions of semantic transport:

$$L_Y \dashv \Theta_Y.$$

It is too early to fix introduction and elimination rules, but this observation considerably narrows the design space. The syntax should somehow distinguish “execute through Y ” from “reason through Y ”, and the adjunction says how these two viewpoints fit together.

6 Two compositions and the problem of context

There is another piece of categorical structure which cannot simply disappear when we pass to syntax. The ambient category is duoidal. Write its two monoidal products schematically as

$$\diamond, \quad \star.$$

One governs the monoid structure from which theories are built; the other governs the measuring interaction and hence the enrichment. They are related by an interchange map of the general form

$$(A \diamond B) \star (C \diamond D) \longrightarrow (A \star C) \diamond (B \star D).$$

What should this mean syntactically? The most conservative answer is that a single undifferentiated cartesian context cannot be expected to remember all of it. We should instead expect two coherent ways of combining assumptions or resources. Schematically there may be context constructions

$$\Gamma \diamond \Delta, \quad \Gamma \star \Delta,$$

together with structural operations corresponding to interchange. Exactly which structural rules each context former admits will depend on the ambient $\mathcal{D}$.

This is where bunched type theory becomes more than a loose analogy. Bunched implication already distinguishes two context-forming operations, traditionally one cartesian and one multiplicative, and permits them to be nested in tree-shaped contexts. Riley’s bunched dependent type theory develops this idea in a setting where linear type formers must coexist with genuine dependency [6]. The point is not that the two products in every duoidal category must literally be interpreted as the two connectives of BI. Rather, Riley demonstrates that retaining two modes of context formation in the syntax is practical, and that dependency across the resulting resource boundary can be mediated by modalities.

Duoidal semantics gives a structural reason to investigate bunched contexts.

A prospective Sweedler calculus should preserve the distinction between the composition used to form theories and the composition used to combine implementation resources, unless there is a semantic reason for those modes to collapse.

7 Riley’s unpointed proposal and Day-convolution resources

Riley’s discussion of unpointed linear types is especially suggestive for our purposes. His main type theory is tailored to parameterised spectra and exploits pointedness strongly. But he asks what happens when the zero object is removed. The resulting picture is much broader. Given a category $\mathcal{C}$, one may form a category PC of parameterised objects, and the special self-adjoint modality of the pointed case is replaced by a longer string of adjoints. Riley then observes that a particularly rich source of models is obtained by taking

$$\mathcal{C} = \text{Psh}(M)$$

for an arbitrary symmetric monoidal category M . Day convolution makes

$$\text{Psh}(M)$$

symmetric monoidal closed, and the further parameterised construction

$$P(\text{Psh}(M))$$

contains M by a fully faithful monoidal inclusion. Riley’s hope is that such a type theory could be used to reason about arbitrary symmetric monoidal categories even when M itself has little additional structure [6, §4.1].

This comes remarkably close to the route we followed in the amortisation chapter. Starting from an ordered monoid or monoidal category C , we formed a presheaf-like category of graded objects

$$C\text{-Set}$$

with Day convolution. In the richer version we then passed to accessible endofunctors

$$[C\text{-Set}, C\text{-Set}]_{\text{acc}}$$

in order to recover the Sweedler theory of monads and runners. Riley is not constructing this endofunctor category, and his modalities should not simply be identified with our runners. The connection lies one level lower: both constructions say that a Day-convolution category built from a resource monoid is not merely a semantic convenience. It is a natural source of a resource-sensitive syntax in which ordinary dependency and monoidal resource composition must coexist.

Riley also points out a difficulty which is illuminating for us. In his unpointed version one can no longer assume

$$1 \otimes 1 \simeq 1.$$

The syntax must therefore remember how copies of the monoidal unit accumulate. What is innocuous in a cartesian context becomes genuine bookkeeping in a resource context. This is exactly the attitude required in our graded semantics: resource structure cannot be erased as though it were ordinary weakening and contraction.

Even more suggestively, Riley proposes that the relevant left-adjoint monadic modality should operate by a kind of locking of variables in the context. A lock controls which dependencies may pass from one structural region into another. We should not simply import that mechanism unchanged, but it offers a concrete answer to a problem which our runner syntax immediately raises: if runner state contains linear, affine or graded resources, which ordinary terms are permitted to depend on it, and when may such dependencies escape?

A useful provisional picture is therefore

$$\Gamma ; \Delta \vdash t : A,$$

where Γ contains ordinary information and Δ contains resource-sensitive information. The semicolon is not intended as settled syntax; it marks a boundary whose structural rules differ on the two sides. Runner execution may transform the resource side,

$$\Delta \mapsto \Delta',$$

while modal or locking operations control how dependency crosses the boundary.

8 Amortisation as the principal worked instance

The cost semantics of Part VI gives the clearest existing example of what this could mean in practice. Let

$$C = (\mathbb{N}, +, 0, \leq)$$

and let

$$\mathcal{V} = C\text{-}\mathbf{Set}$$

carry Day convolution. A representable resource object $y(p)$ gives an operation

$$P_p A := y(p) \otimes A$$

which attaches p units of resource, together with a right-adjoint cost modality

$$M_p A := [y(p), A].$$

This already looks like type-former semantics.

The comparison with λ_{amor} is direct. Its type former

$$[p]A$$

records p units of potential associated with a value of type A , while an indexed cost monad accounts for computation cost [5]. The affine discipline prevents stored potential from being duplicated freely. In the Day-convolution model this is reflected by the fact that a resource-bearing value is genuinely a tensor with a grade:

$$[p]A \rightsquigarrow y(p) \otimes A.$$

The functional queue from Part VI shows why this matters for runners. Its state was schematically

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

An element stored in the first list carries two units of potential. Enqueue performs some immediate work and stores the remaining potential in the runner state. A later expensive dequeue releases that potential when the list is reversed. Semantically,

resource deposited now is retained in runner state and consumed later.

A type system should make that temporal movement of resource visible without requiring us to prove only after execution that the aggregate inequality happened to come out correctly.

This suggests a more informative runner judgement than the bare **Set**-based rule. Schematically one might imagine

$$\frac{\Gamma; \Delta \vdash_T m : A \quad \Gamma; \Sigma \vdash s : S_Y}{\Gamma; \Delta \star \Sigma \vdash_R \text{run}_Y(s, m) : A \otimes S_Y}.$$

The point of the display is not the notation but its structure. Ordinary dependencies live in Γ ; the implementation consumes and transforms explicitly tracked resource context; the resulting R -computation returns an updated resource-bearing state. In a cost model the resource context is graded, whereas in another ambient category the same position might contain linear ownership, a predicate certificate, a smooth parameter or some higher-dimensional concurrency data.

This is where the Riley and λ_{amor} precedents complement one another. λ_{amor} shows how grades and stored potential can be incorporated into types. Riley shows how differently structural pieces of context can coexist with genuine dependency and how modalities can mediate between them. Together they provide a much more concrete guide to an internal language for resource-sensitive runners than either comparison does alone.

9 Changing the ambient world changes the language

The lesson should not be that every example in the tutorial requires a separate type theory. A better architecture is a common runner calculus parameterised by an ambient mode of types and resources. The Sweedler core records theories, tight translations, implementation homs, runner composition and handler transport. Changing $\mathcal{D}$ then changes what may inhabit the value and resource judgements and which additional type formers become available.

A rough table is

ambient setting	additional internal structure
Set	ordinary values and state
Pred_V	refinements and quantitative predicates
C-Set	grades, potential and resource modalities
[pCub, pCub] _{acc}	higher-dimensional/concurrent structure
species settings	symmetry and arity-sensitive resources
[SmoothSet, SmoothSet] _{acc}	smooth dependence and tangent structure
linear/homotopical settings	linear, modal and higher type structure.

The entries in the right-hand column are not claimed to be automatically generated by the duoidal structure. They record additional ambient structure which an adequate internal language should expose rather than erase.

In the predicate setting, for example, a state can carry a judgement

$$P : S \longrightarrow \mathcal{V}$$

expressing an invariant, quantitative bound or robustness margin. In a smooth setting, a controller

$$u : A \times S \longrightarrow TS$$

returns not simply a next state but a tangent vector at the present state. A corresponding syntax must therefore distinguish ordinary values

$$s : S$$

from tangent data

$$v : T_s S.$$

For precubical systems the internal geometry concerns independent transitions and higher-dimensional fillers; for species it concerns symmetry, arity and relabelling. These are not competing accounts of runners. They are different enrichments of the ambient world in which the same runner architecture is interpreted.

Linear Homotopy Type Theory is particularly useful as a precedent at this level. Riley's bunched calculus shows that highly structured parameterised monoidal semantics can acquire an internal language rather than remaining external categorical machinery. Sati and Schreiber's Quantum Monadology provides a related example in which monads and comonads on parameterised linear objects motivate typed effectful syntax intended to live inside a richer linear homotopy type theory [7]. We should not identify either construction with **Run**($\mathcal{D}$). Their significance is methodological: parameterised monoidal semantics, modalities and effectful computation can be made into a usable internal grammar.

10 Execution logic and behaviour logic

There is another distinction which became clearer only in the later parts of the tutorial. An internal runner calculus speaks primarily about the construction and execution of systems. Temporal and extensional properties live at a different level: they speak about complete behaviours.

Earlier we wrote state specifications in a form such as

$$P : S \longrightarrow \mathcal{V}.$$

But temporal properties apply to trajectories, runs or behaviour objects:

$$\Phi : \text{Beh}(S) \longrightarrow \Omega.$$

Safety, guarantee, persistence, response and recurrence are not properties merely of the present state. They quantify over how behaviour unfolds.

The behaviour construction developed later in the tutorial sharpens this separation. Instead of merely hoping that a runner induces some profunctor-like correspondence, we now have the double-functorial picture

$$\text{Beh} : \mathbf{Run}(\mathcal{D}) \longrightarrow \mathbf{CofProf},$$

or the appropriate variant for the ambient setting. Tight theory maps and loose runners therefore induce structured transport at the level of behaviour. The prospective internal language should consequently have two related layers:

execution language how a system is implemented

specification language what its behaviours satisfy.
--

The first is governed directly by theories, Sweedler homs, runners and handlers. The second is organised by behaviour categories and the cofunctor/profunctor structure induced by those same maps.

This is also where Part XI enters the internal-language story. The shadow of **Prof**, and its enriched variants when available, turns suitably composable open behaviour into cyclic behaviour. Thus closing an execution loop is not an unrelated operation added to the syntax at the end. At the behavioural level it is controlled by the shadow/trace structure. In resource-enriched situations a $\mathcal{V}$ -valued shadow may furthermore return not a mere truth value but a cost, weight, probability or other valuation of the closed run.

We should therefore resist trying to put every temporal operator directly into the primitive runner calculus. A cleaner architecture is

$$\text{runner syntax} \longrightarrow \text{behaviour semantics} \longrightarrow \text{specification/trace logic}.$$

The same system can then be studied operationally, extensionally and logically without forcing those viewpoints to coincide.

11 A candidate judgemental core

We can now collect the part of the syntax which seems most strongly suggested by the semantics. Let us write, very schematically,

$$\Gamma; \Delta \vdash A \text{ type}$$

for a type in an ordinary context Γ and an ambient resource context Δ . Depending on $\mathcal{D}$, the context Δ may be absent, cartesian, affine, linear, graded, bunched or geometrically structured. Values have judgements

$$\Gamma; \Delta \vdash a : A.$$

For each theory T there is a computation judgement

$$\Gamma; \Delta \vdash_T m : A.$$

Its primitive computation forms and equations are generated by T .

A tight map

$$f : T \longrightarrow R$$

acts structurally on computations:

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

This is the stateless translation layer.

A runner

$$Y : T \rightsquigarrow R$$

carries state or resource object S_Y . Its elimination principle should internalise

$$TA \otimes S_Y \longrightarrow R(A \otimes S_Y).$$

A schematic rule is therefore

$$\frac{\Gamma; \Delta \vdash_T m : A \quad \Gamma; \Sigma \vdash s : S_Y}{\Gamma; \Delta \star \Sigma \vdash_R \text{run}_Y(s, m) : A \otimes S_Y}.$$

Exactly how $\Delta \star \Sigma$ is formed is ambient-dependent. In a cartesian instance it may collapse to ordinary context extension; in a linear or graded instance it records genuine resource composition.

Runner composition has a structural rule

$$\frac{Y : T \Rightarrow R \quad Z : R \Rightarrow P}{Z \odot Y : T \Rightarrow P},$$

with implementation state combined using the monoidal structure controlling measuring resources. Its computation rule expresses

$$\text{run}_{Z \odot Y} \simeq \text{run}_Z \circ \text{run}_Y.$$

The associativity and unit equations should be inherited from Sweedler-enriched composition.

Handlers finally eliminate the residual theory. If h is an R -handler, there should be a rule schematically of the form

$$\frac{\Gamma; \Delta \vdash_R m : A \quad h : \mathbf{Alg}(R)}{\Gamma; \Delta' \vdash \text{handle}_h(m) : B},$$

where a genuine calculus would record the carrier change which the simplified notation suppresses. Running followed by handling must agree with the induced T -handler $\Theta_Y h$.

Finally, the adjunction

$$L_Y \dashv \Theta_Y$$

suggests runner-indexed modal structure, while the duoidal products suggest more than one context former. These are likely not optional decorations on the calculus: they encode the two characteristic categorical structures which distinguish Sweedler runners from an ordinary algebraic-effects language.

This skeleton is deliberately less committed than a finished typing system but more committed than a list of analogies. Its operations correspond directly to semantic structure already present:

semantic structure	syntactic role
T	effect language
$f : T \rightarrow R$	stateless translation
$\{T, R\}$	universal implementation hom
$Y : T \rightsquigarrow R$	stateful implementation
Sweedler composition	composition of implementations
$L_Y \dashv \Theta_Y$	runner-indexed modal transport
duoidal interchange	interaction of context modes
Beh	passage from execution to behaviour.

12 What remains genuinely open

This reformulation helps distinguish mathematical structure we already possess from design problems which should not be settled by notation alone.

First, we need to decide what an implementation is syntactically. Is a runner a first-class term, a judgement between theories, a modality indexed by a coalgebra, or some combination of these? The Sweedler hom constrains the semantics but does not choose the surface syntax.

Second, state requires a binding discipline. A runner normally has private state which persists between operations but should not automatically be duplicable or visible to the client. In a resource-sensitive setting that state may itself contain potential, ownership or certificates. Riley's bunched contexts and locking modalities give one possible technology for controlling this boundary; algebraic runner calculi give another operational precedent.

Third, the two monoidal structures of the duoidal category must be reflected without over-interpreting them. A two-context or genuinely bunched syntax is attractive, but one must prove that the typing operations correspond to the actual interchange law of the intended models.

Fourth, the runner-indexed adjunction

$$L_Y \dashv \Theta_Y$$

deserves a genuine proof-theoretic treatment. If it produces an adjoint pair of modalities, their introduction, elimination and substitution principles should recover the semantic adjunction rather than merely resemble it.

Fifth, dependency must interact correctly with resources. This is precisely the point at which Riley’s work becomes relevant. In a bunched dependent calculus, types may depend on data while linear structure is still tracked separately. In our setting we additionally want grades, effect theories and implementation state. The interaction among these ingredients is likely to be one of the technically difficult parts of the project.

Finally, the operational and behavioural languages must meet. A proof that a runner implements a theory correctly is not yet a proof of a temporal property of all resulting behaviours. The functorial passage to **CofProf** gives a categorical place where this second layer should live, but its internal proof theory remains to be developed.

13 From theories that run to a language for running theories

We can now look back at the tutorial from a different angle. We began with ordinary comodels and learned to regard a machine not merely as a state object but as an implementation

$$T \rightsquigarrow R$$

between theories. Tight arrows supplied stateless translations; loose arrows supplied stateful realisations; handlers supplied final semantic interpretation. Sweedler theory then revealed that these implementations are organised by universal measuring objects and compose as the homs of an enriched category.

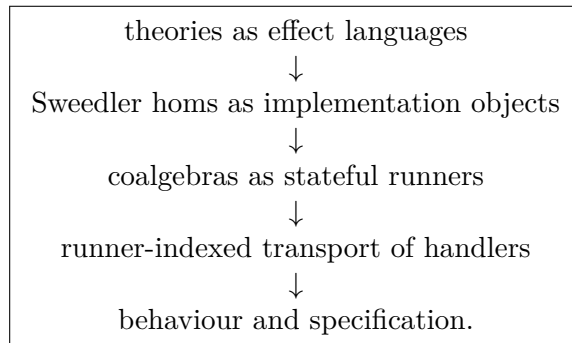
Changing the ambient world repeatedly changed what the implementation state could mean. Predicates turned state into certified state. C -sets turned it into graded resource-bearing state. Precubical sets gave it higher-dimensional concurrency geometry. Species supplied symmetry and many-input/many-output process shape. Smooth settings replaced discrete transition with infinitesimal dynamics. Behaviour categories separated hidden realisation from extensional behaviour, while shadows and wheeled structures explained how open behaviour may be closed and valued.

Seen from the end, these are not merely examples accumulated around a categorical construction. They place increasingly strong demands on its internal language. A user should be able to write an abstract computation, choose or construct an implementation of its effects, run that implementation with controlled resources, compose it with further implementations, interpret the residual theory, and then reason about the behaviour which results.

The existing precedents divide the problem usefully. Algebraic effects already give theories a syntax. Runner calculi already expose resource-managing implementations as programming constructs. λ_{amor} shows that future cost can be stored compositionally in types. Riley’s bunched dependent type theory shows how ordinary dependency and monoidal resource structure can coexist in one language, and his unpointed Day-convolution proposal suggests that this architecture is not

confined to stable homotopy theory. Linear Homotopy Type Theory and Quantum Monadology show that highly structured parameterised monoidal semantics can motivate an internal modal language rather than remaining an external categorical model.

What Sweedler theory contributes is the organising structure tying these clues together:



This makes the research programme considerably more definite than simply asking whether there ought to be some type theory associated with $\mathbf{Run}(\mathcal{D})$. The question is now whether the enriched, duoidal and behavioural architecture can be presented by a usable calculus in which its categorical composition laws become substitution principles and its resource structure becomes context structure.

Prospective programme. Construct an internal language for suitable duoidal settings $\mathcal{D}$ in which monoids provide effect syntax, tight maps provide stateless translations, Sweedler homs provide universal implementation objects, their coalgebras provide stateful runner constructs, Sweedler composition becomes composition of implementations, the adjunctions $L_Y \dashv \Theta_Y$ appear as runner-indexed modal structure, the two monoidal products are reflected by coherent context formers, and passage to behaviour supports a second layer of specification and trace reasoning. Changing $\mathcal{D}$ should change the available resource, refinement, geometric or linear structure without changing this core architecture.

The internal-language question is therefore not an ornament added after the categorical theory has been completed. It may be the point at which the categorical architecture becomes usable as a calculus: not only a theory of machines which run, but a language in which implementations can themselves be composed, typed and reasoned about.

from theories that run, to a language for running theories

References

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.

- [4] I. López Franco and C. Vasilakopoulou. Duoidal categories, measuring comonoids and enrichment. *Theory and Applications of Categories*, 38, 2022. arXiv:2005.01340.
- [5] 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.
- [6] M. Riley. *A Bunched Homotopy Type Theory for Synthetic Stable Homotopy Theory*. PhD thesis, Wesleyan University, 2022.
- [7] H. Sati and U. Schreiber. The Quantum Monadology. arXiv:2310.15735, 2023.