Contextual Sweedler Runners

Part C — histories over injections, coend interaction, and fresh-name abstraction

Part C of the Dependent Sweedler Runners sequence
26 October 2026 — expanded exposition

Abstract

Parts A and B develop stateful runners for operation theories whose commands and continuation worlds depend on a discrete world index. This part asks what changes when worlds are themselves related by renamings and inclusions. Our small example is a machine that can mark an available name or introduce a fresh one. Although each step is already expressible by the dependent polynomial machinery of Part B, the system must now behave coherently when a context is embedded into a larger one. We first explain that coherence through an explicit history comonad. Next, we show why the opposing variance of programs and environments leads naturally to a coend pairing, and prove that the history comonad has a measuring universal property relative to that pairing. Finally, we investigate why a concrete allocator can implement the commands yet fail the usual garbage and symmetry equations of name generation, and how an observation map hides the discrepancy. The result is a worked bridge from dependent runners to contextual, potentially profunctorial interaction; Part D takes up parameterized theories and residual effects.

Status conventions. *Established*: standard mathematics. *Proved*: an argument is given here. *Checked*: additionally verified by computation on random small instances. *Open*: not established.

1 The problem: from indexed worlds to related contexts

A runner, in our usage, is a machine implementing an algebraic theory of operations. Its carrier records state; operations tell us which transitions are possible; and a residual theory, when present, accounts for effects left over from execution. Parts A and B were built to accommodate operations whose result changes the world in which the next operation is well-typed. For example, moving from world n to world $n + 1$ does not require inventing a new notion of machine.

The present problem is subtler. When n counts the names currently in scope, two contexts of different sizes are not isolated sorts. An injection $f : n \hookrightarrow m$ says that the names of the smaller context occur among the names of the larger one. A program using only those names can be renamed along f; a state containing all m names can be restricted to its n-name view. We want a runner to respect both operations.

A running execution. Start with no names and execute

$$\mathsf{new}\ x;\ \mathsf{mark}(x);\ \mathsf{new}\ y;\ \mathsf{mark}(x);\ \mathsf{mark}(y).$$

The first command extends the available context from 0 to 1; the second fresh-name command extends it from 1 to 2. The two marks after the second allocation refer to different in-scope names. Nothing about the exact numerical labels should matter. In particular, embedding the same history into a larger context should not alter the operations it performs on the old names.

This gives the reader a route through the mathematics. Section 2 describes the syntax of available commands. Sections 3–5 classify their contextual state machines. Sections 6–8 use the

program–environment pairing to explain why the comonad is a measuring object, rather than just a convenient formula. Sections 9–10 turn to equations and hidden allocation. The final sections delineate what is established and where the next investigation begins.

2 Background and the discrete benchmark

Parts A and B classified stateful runners for polynomial source theories indexed by discrete worlds. A machine with

$$\mathsf{mark}_i : S_n \to S_n \ (i < n), \qquad \mathsf{new}_n : S_n \to S_{n+1}$$

is a special case. The question here is what survives when world indices carry morphisms representing renaming, forgetting and context extension, and what the characteristic equations of name generation then demand of runners.

3 Binding syntax over injections

The crucial distinction is between *introducing a name in a continuation* and *choosing a concrete storage location*. At the syntactic level, a fresh binder simply extends the context under which its continuation is formed. This is standard territory in the work of Fiore–Plotkin–Turi and in Staton's parameterized theories. Our aim is not to rederive that literature, but to put its basic example into contact with the runner classification of Part B.

We use injections rather than all functions because a context here lists distinct available resource names: an inclusion may add names or reorder them, but does not identify two of them. The larger category of all finite functions remains useful for variables standing for arbitrary substitutable values. These are different disciplines and need not have the same tensor product. Let $\mathbf{I}$ be a skeleton of finite sets and injections, monoidal under disjoint union (not a coproduct in $\mathbf{I}$). Write $\mathcal{P} = [\mathbf{I}, \mathbf{Set}]$, $V = \mathbf{I}(1, -)$, and $\delta X(n) = X(n+1)$, acting on $f : n \hookrightarrow m$ via $f + 1$.

Proposition 3.1 (established)**.** *For Day convolution on $\mathcal{P}$, $\delta \cong [V, -]_{\mathrm{Day}}$.*

Proof. $\mathbf{I}(n, -) \otimes_{\mathrm{Day}} \mathbf{I}(1, -) \cong \mathbf{I}(n+1, -)$, so $[V, X]_{\mathrm{Day}}(n) \cong \mathcal{P}(\mathbf{I}(n+1, -), X) \cong X(n+1)$. □

On $[\mathbb{F}, \mathbf{Set}]$ (all functions) one has instead $\delta X \cong X^V$ for the cartesian product (Fiore–Plotkin–Turi). Over $\mathbf{I}$, context extension is the internal hom of the separating tensor.

Let T be the free monad on $B(X) = V \times X + \delta X$. Write $\mathrm{Hist}(n)$ for finite words of commands from context n ($\mathsf{mark}(i)$ with i below the current context size; new incrementing it), and $\mathrm{end}(h) = n + \#\mathsf{new}(h)$.

Lemma 3.2 (proved)**.** *$T(X)(n) \cong \coprod_{h \in \mathrm{Hist}(n)} X(\mathrm{end}\, h)$, naturally in X; in particular $T(1)(n) \cong \mathrm{Hist}(n)$.*

Proof. All operations are unary, so a term is a command word followed by a leaf in X at the context reached. □

4 The discrete dependent runner

Before accounting for renamings, suppress every nonidentity map between contexts. A world is then just a natural number. At n, a mark command continues in the same world; a fresh-name

command continues in the next one. Each command has exactly one continuation, so its history is an ordinary typed word rather than a branching tree.

This is an example already handled in Part B. Recording it matters because it locates the new work: the type-changing arrow $S_n \to S_{n+1}$ is *not* the difficulty. The difficulty is requiring every such arrow to be compatible with a whole category of context embeddings. Over the discrete worlds $\mathbb{N}$, the command polynomial is $Q(X)_n = \coprod_{i<n} X_n + X_{n+1}$. By the free-polynomial classification of Part B (residual: the path monad of $(\mathbb{N}, \leq)$), runners are families S with $m_{i,n} : S_n \to S_n$ and $\nu_n : S_n \to S_{n+1}$, and the cofree classifier is

$$(CY)_n = \prod_{h \in \mathrm{Hist}(n)} Y_{\mathrm{end}(h)}.$$

5 Contextual cofree histories

What should a state tell us about the future? For every legal history h, starting in its current context, it determines a state at the end of that history. The cofree-history object packages all these hypothetical future states in one family. This is analogous to the behaviour-tree construction of the earlier Moore examples, except that the endpoint sort varies with the history.

A context embedding acts on this package by asking only about the histories that come from the smaller context. The larger context has extra commands using its extra names, but restriction is not obliged to recover those commands. This simple observation explains why the construction is contravariantly functorial even though the alphabet of commands grows. Let $\mathcal{S} = [\mathbf{I}^{\mathrm{op}}, \mathbf{Set}]$: runtime states vary contravariantly, restriction forgetting names outside a smaller context. An injection $f : n \hookrightarrow m$ transports $h \in \mathrm{Hist}(n)$ to $f_* h \in \mathrm{Hist}(m)$ (renaming marks, extending f by one at each new), with induced endpoint injection $f_h : \mathrm{end}(h) \hookrightarrow \mathrm{end}(f_* h)$. For $Y \in \mathcal{S}$ set

$$(CY)(f)(\varphi)_h = Y(f_h)\big(\varphi_{f_* h}\big).$$

Proposition 5.1 (proved)**.** *This defines a comonad C on $\mathcal{S}$. Its coalgebras are exactly presheaves S with transitions $m_{i,n}$, ν_n satisfying, for every $f : n \hookrightarrow m$ and $i < n$,*

$$S(f)\, m_{f(i),m} = m_{i,n}\, S(f), \qquad S(f+1)\, \nu_m = \nu_n\, S(f).$$

Sketch. Transport of histories respects identities, composition and concatenation, giving functoriality; counit and comultiplication are evaluation at the empty history and concatenation. A coalgebra is determined by its one-step transitions; naturality of the coalgebra map is equivalent to the displayed equations. □

We call such coalgebras **I**-*runners*. The canonical heap model (Example 9.5(a)) is one (checked: 3000 random programs, injections and states).

6 Variance: commands and comodels

At this point one might try to apply the discrete polynomial formula directly to presheaves of runtime states. This is where a genuine mismatch occurs. Commands become *more* plentiful when names are added, whereas the restriction of an environment discards information. A command mentioning a newly added name has no canonical meaning in the smaller context.

For the machine itself, however, we need a *choice of successor state for each available command*, which is a product rather than a coproduct. Products restrict by projecting away answers

to the extra commands. This explains the contrast between Q and F below; it is the familiar algebra/comodel asymmetry now visible as a variance issue.

Proposition 6.1 (proved)**.** *The command polynomial* $(QS)_n = \coprod_{i<n} S_n + S_{n+1}$ *has no canonical command-preserving contravariant action on* $\mathcal{S}$*. The one-step comodel functor*

$$(FS)_n = \prod_{i<n} S_n \times S_{n+1}, \qquad (FS)(f)\big((s_i)_{i<m}, s'\big) = \big((S(f)s_{f(i)})_{i<n},\, S(f+1)s'\big)$$

does, **I***-runners are exactly* F*-coalgebras in* $\mathcal{S}$*, and* C *is the cofree comonad on* F*.*

Proof. For Q, a summand $\mathsf{mark}(j)$ with $j \notin \operatorname{im} f$ has no command-preserving image at context n. For F, restriction projects onto the components indexed by $\operatorname{im} f$. Limits in $\mathcal{S}$ are pointwise, so the terminal-sequence construction of the cofree coalgebra is computed in $\mathbf{Set}^{\mathbb{N}}$ and inherits the action, giving Proposition 5.1. □

Commands (the Σ-side) live covariantly with the syntax; comodel structure (the Π-side) lives contravariantly with the state. This is the model/comodel asymmetry reappearing at the level of context variance.

7 Change of context

There are two ways to ask about change of context. The modest one is whether the behaviour assigned to a larger context restricts to that of a smaller context; the presheaf maps of C answer yes. The stronger question is whether this restriction loses no information. A proper inclusion adds new commands, so in general the answer must be no.

The following proposition makes that distinction precise. It is deliberately a theorem about restriction maps, not yet a general Beck–Chevalley theorem for fibred Sweedler adjunctions. It echoes the tree-lifting questions of Part B without assuming those comparisons are identical.

Proposition 7.1 (proved)**.** *For* $f : n \hookrightarrow m$*, the following are equivalent: (i)* f *is a bijection; (ii)* $f_* : \mathrm{Hist}(n) \to \mathrm{Hist}(m)$ *is a bijection; (iii)* $(CY)(f) : (CY)_m \to (CY)_n$ *is invertible for* every $Y \in \mathcal{S}$*.*

Proof. (i)⇒(ii),(iii): f_* and all f_h are bijections. (ii)⇒(i): length-one histories $\mathsf{mark}(i)$ biject only if f does. (iii)⇒(ii): take Y constant at 2; then $(CY)(f)$ is precomposition $2^{\mathrm{Hist}(m)} \to 2^{\mathrm{Hist}(n)}$ with f_*, invertible only if f_* is bijective. □

Remark 7.2. The statement is uniform in Y: for particular Y (e.g. singleton-valued) $(CY)(f)$ may be invertible accidentally. Reading f as a morphism from the one-world signature at n to that at m ($\mathsf{mark}_i \mapsto \mathsf{mark}_{fi}$, $\mathsf{new} \mapsto \mathsf{new}$), Proposition 7.1 is the instance of the comonad-side criterion from our earlier working notes (the cofree comparison is invertible for all Y iff the signature morphism is cartesian, at viable worlds). It should be contrasted with the monad-side tree-lifting criterion of Part B (tree-convexity): the two sides have different Beck–Chevalley classes. Here the failure along proper injections is not a defect: the non-invertible restrictions are precisely the presheaf action of C.

8 The program–environment pairing

The next question is why the history comonad should be called *Sweedler-like*. Until now it has been constructed directly from transitions. To connect it with the theory being executed, we should pair a program with a state in which it can run.

The program presheaf is covariant: an injection renames each free name into a larger context. The environment presheaf is contravariant: an environment in a larger context can be viewed through the names of a smaller one. These two actions have opposite variance, so the correct elementary pairing is a coend. It identifies a program renamed into a larger context together with a large environment with the original program together with the restricted environment.

This pairing is a concrete instance of the profunctorial theme that has also surfaced in our work on behaviour and observation. But we do not assume a general two-sided Sweedler theory: the point of the next calculations is to see exactly how much universal structure this one example supports. For $P \in \mathcal{P}$, $S \in \mathcal{S}$ write $P \otimes_{\mathbf{I}} S = \int^n P(n) \times S(n)$. A context-independent interpreter $\mathrm{run}_n : P(n) \times S(n) \to Z$ factors through the coend iff $\mathrm{run}_m(P(f)p, s) = \mathrm{run}_n(p, S(f)s)$. For results depending on context we use the following.

Definition 8.1. For an $\mathbf{I}$-runner S, define $\langle -, - \rangle : T(X) \otimes_{\mathbf{I}} S \to X \otimes_{\mathbf{I}} S$ by executing the command word of $p \in T(X)(n)$ on $s \in S(n)$ and returning $[x, s']$, where $x \in X(n+k)$ is the leaf and $s' \in S(n+k)$ the final state.

Lemma 8.2 (proved; checked)**.** *$\langle -, - \rangle$ is well defined: $\langle T(X)(f)p, s \rangle = \langle p, S(f)s \rangle$ for $f : n \hookrightarrow m$, $p \in T(X)(n)$, $s \in S(m)$.*

Proof. Induction on p using the runner equations; at the leaf, $[X(f+k)x, s'] = [x, S(f+k)s']$ is a coend identification. □

Proposition 8.3 (proved)**.** *For $Y \in \mathcal{S}$ define $D(Y)(n) = \mathrm{Nat}_{X \in \mathcal{P}}\big(T(X)(n),\, X \otimes_{\mathbf{I}} Y\big)$. Then $D(Y) \cong CY$, naturally in Y.*

Proof. By Lemma 3.2 and Yoneda (the functor $X \mapsto X(e)$ is represented by $\mathbf{I}(e, -)$),

$$D(Y)(n) \cong \prod_{h \in \mathrm{Hist}(n)} \mathrm{Nat}_X\big(X(\mathrm{end}\, h), X \otimes_{\mathbf{I}} Y\big) \cong \prod_h \mathbf{I}(\mathrm{end}\, h, -) \otimes_{\mathbf{I}} Y \cong \prod_h Y(\mathrm{end}\, h),$$

the last step by co-Yoneda. □

Thus the explicit history comonad is the transpose of the program monad against the coend pairing. The transpose also respects monad and comonad structure, as follows.

The transpose formula is the central conceptual turning point. For each possible command history, the representing object stores the response to the continuation placed after that history. The naturality in the continuation presheaf ensures that these responses depend only on the history and endpoint, not on an arbitrary presentation of the leaf. By Yoneda, giving all such responses is exactly giving a family indexed by histories.

To upgrade this objectwise representation to a measuring statement, we must ask how it treats *return* and *substitution of continuations*. Those operations are the unit and multiplication of the program monad; their counterparts on histories are the empty history and concatenation.

Definition 8.4 (Coend-relative measuring interaction)**.** Let $D = (D, \varepsilon^D, \Delta^D)$ be a comonad on $\mathcal{S}$. A *measuring interaction* is a family, natural in $X \in \mathcal{P}$ and $Y \in \mathcal{S}$,

$$\theta_{X,Y} : T(X) \otimes_{\mathbf{I}} D(Y) \longrightarrow X \otimes_{\mathbf{I}} Y,$$

satisfying

$$\theta_{X,Y} \circ (\eta_X \otimes_{\mathbf{I}} \mathrm{id}_{DY}) = \mathrm{id}_X \otimes_{\mathbf{I}} \varepsilon^D_Y,$$
$$\theta_{X,Y} \circ (\mu_X \otimes_{\mathbf{I}} \mathrm{id}_{DY}) = \theta_{X,Y} \circ \theta_{TX,DY} \circ (\mathrm{id}_{T^2X} \otimes_{\mathbf{I}} \Delta^D_Y).$$

In the second law the middle arrow has type $T^2X \otimes_{\mathbf{I}} D^2Y \to TX \otimes_{\mathbf{I}} DY$.

The first law says that a program consisting only of a returned value makes no operational demand: the environment is observed through the comonad counit. The second law says that executing an outer history and then its substituted continuation is the same as executing their concatenation, with the comonad supplying the successive views needed by the two executions. Their types are worth retaining because they keep the two sides of the pairing separate.

Theorem 8.5 (Two-sided measuring representation for free contextual histories)**.** *For every comonad D on $\mathcal{S}$, coend transposition restricts to a natural bijection*

$$\{\textit{measuring interactions } T(-) \otimes_{\mathbf{I}} D(-) \to - \otimes_{\mathbf{I}} -\} \;\cong\; \mathrm{Comon}(\mathcal{S})(D, C).$$

Thus C is terminal among comonads equipped with a measuring interaction with this particular free contextual unary monad T.

Proof. By Proposition 8.3, an interaction θ has a unique transpose $\phi : D \Rightarrow C$. For $d \in DY(n)$ and a history h starting at n, the component $\phi_Y(d)_h \in Y(\mathrm{end}\, h)$ is obtained by applying θ to the generic program h followed by the representable continuation at $\mathrm{end}\, h$, then using co-Yoneda. In particular, the unit law says that the component at the empty history is $\varepsilon^D_Y(d)$, precisely $\varepsilon^C \phi = \varepsilon^D$.

The terms of T^2X are indexed by pairs (h, g) of composable histories. The monad multiplication sends (h, g) to concatenation hg. Evaluating the multiplication measuring law on the generic nested term (h, g) therefore says, on the left, $\phi_Y(d)_{hg}$, and on the right, the (h, g)-component of

$$C(\phi_Y) \circ \phi_{DY} \circ \Delta^D_Y(d).$$

Since $\Delta^C_Y(\varphi)_h(g) = \varphi_{hg}$, equality for all such generic terms, followed by naturality in the continuation presheaf, is exactly

$$\Delta^C \circ \phi = C\phi \circ \phi_D \circ \Delta^D.$$

Conversely, the canonical interaction of C evaluates a program history against its matching behaviour component and returns the continuation leaf paired with the resulting state. Its empty-history and concatenation laws give the two measuring axioms. Composing this interaction with a comonad morphism $D \to C$ preserves both axioms. Coend transposition makes the two constructions inverse. □

Remark 8.6. The theorem concerns the coend pairing between *different* functor categories, not the ordinary Sweedler hom of monads and comonads on one ambient category. It requires neither an abstract two-sided Sweedler construction for arbitrary monads nor a residual interaction law. Part D takes up residual interactions for this pairing.

9 The equations of name generation

The universal property concerns the *free* theory: it permits all legal sequences and imposes no identifications between distinct command histories beyond the monad laws. Realistic accounts of fresh names usually demand more. If a generated name is unused, an external observer should not

detect its allocation; two fresh names should be exchangeable; and generating a fresh name should not disturb an operation on an older name.

Those intuitions become three equations. The crucial issue is what it means for an implementation to satisfy them. The coend gives a precise test: run the two terms and compare their resulting continuation–environment pairs. This comparison can be stricter than observational equivalence of programs, because the environment may retain hidden allocation details. In de Bruijn-level form, with $\iota_n : n \hookrightarrow n+1$ and τ_n exchanging the last two names of $n+2$:

$$\begin{aligned}
\text{(G) garbage:} &\quad \mathsf{new}\big(T(X)(\iota_n)\,t\big) = t && (t \in T(X)(n)),\\
\text{(S) swap:} &\quad \mathsf{new}\,\mathsf{new}\,t = \mathsf{new}\,\mathsf{new}\,T(X)(\tau_n)\,t && (t \in T(X)(n+2)),\\
\text{(M) commutation:} &\quad \mathsf{mark}(i);\mathsf{new};t = \mathsf{new};\mathsf{mark}(i);t && (i < n,\ t \in T(X)(n+1)).
\end{aligned}$$

The two sides of (G) end in different contexts, so validation is stated in the coend.

Definition 9.1. An $\mathbf{I}$-runner S *strictly validates* $l = r$ if $\langle l, s\rangle = \langle r, s\rangle$ for all instances and all s.

Proposition 9.2 (proved; checked)**.** *An* $\mathbf{I}$*-runner strictly validates*

$$\textit{(G)} \iff S(\iota_n)\,\nu_n = \mathrm{id}_{S_n}; \qquad \textit{(S)} \iff S(\tau_n)\,\nu_{n+1}\nu_n = \nu_{n+1}\nu_n; \qquad \textit{(M)} \iff \nu_n\, m_{i,n} = m_{i,n+1}\,\nu_n.$$

Proof. Sufficiency, by Lemma 8.2:

$$\begin{aligned}
\langle \mathsf{new}\,T(X)(\iota)t,\ s\rangle &= \langle T(X)(\iota)t,\ \nu s\rangle = \langle t,\ S(\iota)\nu s\rangle = \langle t, s\rangle,\\
\langle \mathsf{new}\,\mathsf{new}\,T(X)(\tau)t,\ s\rangle &= \langle T(X)(\tau)t,\ \nu\nu s\rangle = \langle t,\ S(\tau)\nu\nu s\rangle = \langle t, \nu\nu s\rangle = \langle \mathsf{new}\,\mathsf{new}\,t, s\rangle,\\
\langle \mathsf{mark}(i);\mathsf{new};t,\ s\rangle &= \langle t,\ \nu\, m_i s\rangle = \langle t,\ m_i\,\nu s\rangle = \langle \mathsf{new};\mathsf{mark}(i);t,\ s\rangle.
\end{aligned}$$

Necessity: take generic instances and use co-Yoneda, $\mathbf{I}(e,-)\otimes_{\mathbf{I}} S \cong S(e)$, $[g, s'] \mapsto S(g)s'$. For (G), $X = \mathbf{I}(n,-)$ and $t = \mathrm{return}(\mathrm{id}_n)$: the two sides become $S(\iota)\nu s$ and s. For (S), $X = \mathbf{I}(n+2,-)$ and $t = \mathrm{return}(\mathrm{id})$: they become $\nu\nu s$ and $S(\tau)\nu\nu s$. For (M), $X = \mathbf{I}(n+1,-)$ and $t = \mathrm{return}(\mathrm{id})$: they become $\nu m_i s$ and $m_i \nu s$. □

(G) is an effect-masking condition (an allocation not exposed to the continuation is unobservable); (S) is a symmetry condition on distinct fresh names; (M) relates allocation to the dynamics of existing names. They are independent requirements.

9.1 Equations as constraints on future behaviour

The preceding proposition characterises when a *particular machine* obeys the three equations. We can now ask a stronger, universal question: can we classify *all* machines satisfying them by a comonad, just as C classifies all unconstrained contextual machines? This is the comodel counterpart of quotienting an algebraic source theory by equations. The answer is yes for the specified unary equations, provided we test the equations not only at the initial observation but throughout every possible future.

There is a small but essential point here. A behaviour $\varphi \in (CY)(n)$ is a family of observations indexed by histories. Equality of its values immediately after two command sequences is not enough: a later command could distinguish the two resulting states. We therefore test the equations after arbitrary history prefixes *and* with arbitrary well-typed continuation experiments. The comparison across different endpoint contexts is always made using the restriction maps of Y, or equivalently using the coend relation of the preceding section.

Let $E = \{\mathrm{G}, \mathrm{S}, \mathrm{M}\}$ be the typed schematic program equations displayed above. A *contextual test* of an equation $l = r$ consists of a legal history prefix, an instance of its parameter and context variables, and a continuation/observation experiment with matching result types. Via the canonical measuring interaction of C, the two sides produce observations in Y (with endpoint comparisons performed by reindexing). Say that $\varphi \in (CY)(n)$ *satisfies E hereditarily* if every such test gives equal observations. Define

$$(C_E Y)(n) = \{\varphi \in (CY)(n) \mid \varphi \text{ satisfies every contextual test of } E\}.$$

This definition deliberately includes continuation tests: merely requiring $\varphi_h = Y(\iota)\varphi_{h\cdot\mathsf{new}}$, and the analogous immediate swap and commutation equalities, would not suffice to ensure closure under all future observations.

Proposition 9.3 (Strict equational behaviour classifier)**.** *For the free contextual unary theory and the three specified equations E, the inclusions $C_E Y \hookrightarrow CY$ are natural in Y and form a subcomonad $C_E \hookrightarrow C$. Its coalgebras are exactly the* **I***-runners whose transitions satisfy the three laws in Proposition 9.2. Moreover, relative to the coend pairing, for every comonad D on $\mathcal{S}$ there is a natural bijection*

$$\{\ \textit{measuring interactions of } T \textit{ with } D \textit{ that validate every equation of } E\ \} \cong \mathrm{Comon}(\mathcal{S})(D, C_E).$$

The validation on the left is strict: it uses equality in the coend, not equality merely after an observational abstraction.

Proof. Transporting a contextual test along an injection gives another contextual test, so hereditary satisfaction is preserved by the presheaf restrictions. Applying an observation map $Y \to Y'$ preserves test equalities; hence C_E is a subfunctor of C. The empty-history counit restricts automatically. Every future shift of a hereditarily satisfying behaviour again satisfies all tests: a test after shifting by h is simply a test after the longer prefix h. The same argument, with the tests in the continuation slot, shows that history concatenation sends $C_E Y$ into $C_E(C_E Y)$. Thus the counit and comultiplication of C restrict and satisfy the comonad laws.

For a C-coalgebra $\gamma : S \to CS$, the h-component of $\gamma(s)$ is the state obtained by executing h from s. The image of γ lies in $C_E S$ precisely when the two sides of every equation remain indistinguishable after arbitrary prefixes and continuations. Because the continuation may be the empty history, necessity gives the state-level identities of Proposition 9.2. Conversely, those identities hold at every state, hence after each prefix, and their compatibility with subsequent transitions proves all continuation tests.

Finally, the measuring representation of Theorem 8.5 sends any interaction to its unique comonad morphism $D \to C$. Validity of the equations, including all continuation tests, says exactly that the image consists of hereditary E-behaviours. Such a morphism therefore factors uniquely through $C_E \hookrightarrow C$; conversely a morphism into C_E gives a measuring interaction validating E. □

Remark 9.4 (Scope of the proposition). This result imposes equations on the *free unary* new/mark theory. It does not yet identify C_E with the measuring comonad of Staton's full restriction-and-equality theory: equality testing has two continuations, and his parameter substitutions introduce additional structure. Those are separate problems, not consequences of this quotient construction. Nor do the strict equations characterise product-store models; the constant-presheaf example below already shows otherwise.

Example 9.5. In each case restriction along f acts on environments by precomposition.

(a) *Concrete heap.* $S_n = \{(L, M, \rho) \mid L \subseteq \mathbb{N} \text{ finite},\ M \subseteq L,\ \rho : n \hookrightarrow L\}$; m_i adds $\rho(i)$ to M; ν adds $a = \min(\mathbb{N} \setminus L)$ to L and appends it to ρ.

(b) *Heap modulo permutations.* As (a), quotiented by bijections of $\mathbb{N}$ acting on (L, M, ρ).

(c) *Local store.* For a set A, $a_0 \in A$, $g : A \to A$: $S_n = A^n$, ν appends a_0, m_i applies g to the i-th entry.

(d) *Constant presheaf.* $S_n = K$ with identity restrictions, $\nu = \mathrm{id}$, $m_i = u$ for a fixed $u : K \to K$.

Model	(G)	(S)	(M)
(a) concrete heap	no	no	yes
(b) heap modulo permutations	no	yes	yes
(c) local store	yes	yes	yes
(d) constant presheaf	yes	yes	yes

In (a), $\mathsf{new}; \mathsf{new}; \mathsf{mark}(0)$ and $\mathsf{new}; \mathsf{new}; \mathsf{mark}(1)$ mark different locations; restriction forgets the environment but not the allocation, so (G) fails (the heap grows), and (S) fails because the marked locations differ. Quotienting by permutations restores (S) but not (G): heaps of different sizes remain distinct. Model (d) shows that the equations do not force a product store: strict models need not consist of independently stored values per name.

Remark 9.6. Two obstructions should not be conflated. There is no equivariant choice of fresh atom; this concerns the action of permutations on locations and does not affect runners of the free theory, since the deterministic allocator in (a) is natural in context renamings. Separately, the equations (S) and (G) impose fresh-name symmetry and hiding, which particular models such as (a) fail.

10 Validation up to observation

The heap examples expose a familiar phenomenon. A counter-based or least-unused-location allocator can execute every command correctly and be natural in syntactic renamings, yet record an allocation that an abstract program is supposed to treat as invisible. The failure is not a failure of context naturality. It is a mismatch between the *raw implementation state* and the *information a client is allowed to observe*.

We therefore separate an implementation from an observation of it. An observation map should preserve execution and may discard private heap information. Equations can then hold after applying that map even when they fail as literal equalities of concrete states. This is a deliberately limited precursor to the logical relations used for resource abstraction in Cambria, not a replacement for those relations. Concrete implementations retain hidden allocations. We formulate the weakest useful notion.

Definition 10.1. A morphism of $\mathbf{I}$-runners $\pi : S \to S'$ is a natural transformation commuting with m_i and ν. *S validates $l = r$ up to π* if $(X \otimes_{\mathbf{I}} \pi)\langle l, s\rangle = (X \otimes_{\mathbf{I}} \pi)\langle r, s\rangle$.

Since runner morphisms commute with execution, $(X \otimes_{\mathbf{I}} \pi)\langle p, s\rangle = \langle p, \pi s\rangle'$; hence if S' strictly validates $l = r$, then S validates it up to π.

Proposition 10.2 (proved; checked)**.** *Let S be the concrete heap (a) and $S'_n = \mathrm{Sub}(n)$ the presheaf of subsets (restriction by preimage; m_i adds i; ν is the identity on subsets), which is the local store (c) with $A = \{0, 1\}$, $a_0 = 0$, $g \equiv 1$. Then*

$$\pi_n(L, M, \rho) = \rho^{-1}(M)$$

is a morphism of **I***-runners. Consequently the concrete heap validates (G), (S) and (M) up to π.*

Proof. Naturality: $(\rho f)^{-1}M = f^{-1}\rho^{-1}M$. Marking: $\rho^{-1}(M \cup \{\rho(i)\}) = \rho^{-1}M \cup \{i\}$ by injectivity. Allocation: the fresh $a \notin L \supseteq M$, so $(\rho, a)^{-1}M = \rho^{-1}M$. Random check: 6000 instances of (G) and (S) differ on raw states after scope exit in 4097 cases, and agree in all cases after π. □

Remark 10.3 (Relation to Cambria)*.* Liell-Cock and Staton establish the garbage law for a counter-based fresh-name handler not on states but via a parametricity relation $\{(i, i+1)\}$ between handler states. Proposition 10.2 is a functional special case of this idea. In general one should not expect a quotient of a single carrier to suffice: a parametricity relation relates different implementations, and hiding may relate states over different worlds. The natural candidate is a proof-relevant, world-indexed relation $\mathcal{R}_{n,m} \subseteq S_n \times S'_m$ closed under restriction, marking and allocation. This is close to the certified-runner and must/may machinery developed in the arrow topos, and suggests a synthesis of contextual runners, relational certification, and hiding of generated resources.

11 How this fits the existing literature and Parts A–B

The preceding calculations should not be read as a new discovery of variable binding or local allocation. Those theories are well developed. Our more specific question is what their programs look like when regarded as source theories for *stateful interactions*, and how a comonadic classifier keeps track of contexts, histories, and observation.

Part A supplied concrete typed runners; Part B supplied dependent polynomial classification and base-change tests. Part C adds the non-discrete context category and demonstrates, for a free unary theory, that the history classifier is also the measuring object for a covariant/contravariant pairing. Part D starts from this pairing and asks what a residual monad can retain about an execution.

- Fiore–Plotkin–Turi: abstract syntax with binding over finite contexts. For names denoting distinct resources, **I** is the standard base (Stark; Plotkin–Power local state; nominal sets).
- Staton's parameterized algebraic theories: equations for name generation, including the axiomatisation of freshness.
- Kammar–Levy–Moss–Staton: a possible-worlds monad for full ground references with allocation and hiding; Proposition 9.2 isolates why hiding is needed (condition (G)).
- Van Rooij–Kammar–Lindley–Matache (2026): that monad is not finitary; they construct an adequate finitary submonad, towards an equational presentation.
- Liell-Cock–Staton, *Cambria* (POPL 2027): parametrized handlers with a parametricity theorem giving resource abstraction; the denotational, categorical structure of parametrized handlers is listed as future work.

The contribution sought here is not a new syntax or allocator but a Sweedler-style classification of stateful interpretations that preserves contextual structure, together with a notion of validation that admits hidden state.

12 From contextual histories to residual runners

Part D, *Residual Sweedler Runners over Contexts*, keeps the pairing of covariant programs with contravariant state presheaves and lets a residual monad record the outcome of execution. It begins with the coend representation theorem for accessible monads and recovers the history classifier above as its unary case. Dynamic threads explain why the residual matters. A thread can stop without invoking a continuation, so a runner required to return a continuation has no states. A multiset residual can record the actions of both threads. To preserve causal dependencies, one must also account for which thread IDs are available at each continuation.

Problem 1 (Contextual residuals)**.** Find a residual from which one can recover the causal dependencies at a continuation of an open thread program. Compare it with the labelled posets with holes used in Part D and with the treatment of hidden names in local state.

Problem 2 (Relational validation)**.** Define world-indexed, proof-relevant relations between runners compatible with restriction and allocation; show that they recover Proposition 10.2 and Cambria's counter argument as instances.

Problem 3 (Strict models)**.** Classify **I**-runners validating (G), (S) and (M). Model (d) shows that additional locality or independence assumptions are needed for any characterisation by product stores.

Problem 4 (Richer contexts)**.** Replace **I** by structured contexts—document shapes and edits (Denicek), typed graph regions, Peirce Beta configurations—where reference transport may be partial, forcing spans or partial maps into the base.

13 Results and limits

The ledger is included to separate standard background from calculations established in this part and from conjectural generalisations. In particular, the measuring theorem is proved only for this free unary contextual syntax: it should not be presented as a general representation theorem for all GATs, parameterized effects, or residual runners.

Summary of the narrative

The passage from Part B to Part C is not about allowing a command to change its world—we already knew how to do that. It is about *relating different descriptions of a world* and requiring execution to respect those descriptions. Injections relate name contexts; the cofree comonad records every possible future; the coend pairs syntax with restricted environments; and the measuring theorem shows that these descriptions fit a common universal pattern. Name-generation equations then test which implementations expose too much state. This is the point of contact with resource abstraction and the reason to look next at Staton's non-free parameterized theories.

14 Ledger

Claim	Status
$\delta \cong [V, -]_{\mathrm{Day}}$; $T(X)(n) \cong \coprod_h X(\mathrm{end}\, h)$	established / proved
Discrete runners and cofree classifier	proved (Part B)
Contextual cofree-history comonad and its coalgebras	proved
Q does not extend to $\mathcal{S}$; F does, with C its cofree comonad	proved
Invertibility of $(CY)(f)$ for all Y iff f bijective	proved
Coend pairing well defined	proved; checked
History comonad is the transpose of T against the coend	proved
Coend measuring universal property for free unary T	proved
Residual coend representation for accessible monads	taken up in Part D
Characterisation of (G), (S), (M)	proved; checked
Strict equational subcomonad C_E (free unary theory)	proved
Four-model table	proved; checked (a),(c)
Heap validates the equations up to π	proved; checked
Relational validation recovering Cambria	open

References

[1] M. Fiore, G. Plotkin, D. Turi. Abstract syntax and variable binding. *LICS* 1999.

[2] S. Staton. Instances of computational effects: an algebraic perspective. *LICS* 2013.

[3] O. Kammar, P. B. Levy, S. K. Moss, S. Staton. A monad for full ground reference cells. *LICS* 2017. arXiv:1702.04908.

[4] O. van Rooij, O. Kammar, S. Lindley, C. Matache. Finitary semantics for full ground local state. arXiv:2608.21271, 2026.

[5] J. Liell-Cock, S. Staton. Cambria: resource abstraction for parametrized algebraic effects and handlers. *POPL* 2027. arXiv:2608.27798.

[6] D. McDermott, E. Rivas, T. Uustalu. Sweedler theory of monads.

[7] *Elementary Dependent Sweedler Runners: A Gallery of Small Machines*, Part A (this series).

[8] *The Structure Behind Dependent Runners: Refined Worlds, Polynomials, and Change of Context*, Part B (this series).