Getting to Grips with Sweedler Runners for Systems

Temporal Interlude: Execution, Synchronization, and Quantitative Verification

From free observation theories to temporal sheaves and marked-event logic

October 2026, final corrections

Aim. We want to understand what it would mean to run a program in a world where behaviour occupies time. A source operation may require several target observations; each observation may consume an actual duration. We begin with ordinary stateful Sweedler runners, unfold them into histories, compose their histories, and attach quantitative certificates and monitors. Finally we compare the interval-sheaf semantics of [2] with the more local temporal topos of [1]. An alternating-duration oscillator provides an exact internal bounded-response calculation.

Foundational boundary. The constructions use explicit free-monad runners and timed physical realizations. We do *not* assume that all accessible endofunctors on an arbitrary Grothendieck topos possess the requisite Sweedler measuring objects. The general existence theorem remains an independent problem.

Contents

1 Why time changes the story **2**

2 External endofunctors and internal operations **4**

3 Free observation theories and stateful runners **5**
3.1 A one-read implementation 6

4 One-read runners unfold into discrete interval sheaves **7**
4.1 Connecting two histories 7

5 Two reads per source operation: phase matters **8**
5.1 From implementing trees to event-clocked machines 9
5.2 Substitution still agrees with execution 10

6 Time distortion and amortised certificates **10**
6.1 A small example before the composition theorem 11
6.2 Why certificates compose 11

7 Transporting a bounded-response specification **12**

8 Timed monitors: turning deadlines into state safety **13**
8.1 The monitor as another temporal machine 13

9 A genuinely continuous-time runner **14**
9.1 A sampler that chooses its next delay 15
9.2 Continuous suspension and its interval sheaf 15
9.3 The delayed reading acquires a real deadline 16

10 Real-duration amortisation and non-Zeno obligations **17**
10.1 An upper bound is not a lower bound 17

11 Two interval toposes, not one **17**
11.1 Why the distinction is real 18

12 Exact sheafification of the oscillator **19**
12.1 What happens to punctual events? 20

13 The clock object and the sharp bounded-response calculation **20**
13.1 Marking a request without a zero-duration state 21
13.2 Looking for a response witness 21
13.3 Comparing the answer with a timed monitor 22
13.4 Towards a genuinely clock-indexed response proposition 22

14 Composition with monitors after temporal sheafification **23**

15 What is proved, and what remains **23**

References **25**

1 Why time changes the story

In the earlier parts of the tutorial a runner was a way to implement the operations of one theory in another, retaining a residue of internal state. One could ask a machine for a Boolean reading, update a memory bit, and carry on. But how long did that implementation take? Was the response delivered before a deadline? Could a computation stop midway through an operation? These questions have no answer until the runner is equipped with a temporal realization.

A recurring theme of this tutorial has been the relationship between a system and its behaviour: between the operations by which we interact with it and the ways it can evolve and respond. There is a closely related line of work by Patrick Schultz and David Spivak, together with Christina Vasilakopoulou in Dynamical Systems and Sheaves [2], and developed further by Schultz and Spivak in Temporal Type Theory [1]. Their starting point is to treat behaviour over an interval of time as a mathematical object in its own right. A behaviour can be restricted to a shorter interval, and compatible behaviours can be assembled into longer ones. Sheaf theory provides a natural language for these operations. It allows continuous dynamics, discrete transitions, and combinations of the two to be studied within a common categorical framework.

The two papers develop this idea in somewhat different directions. *Dynamical Systems and Sheaves* is particularly concerned with machines and their interconnection. A machine is described through its possible histories of inputs, outputs, and internal states; compatible histories can be joined at their shared boundaries, and machines can be composed using spans and pullbacks. *Temporal Type Theory* develops a logic for reasoning about behaviour over time, including temporal

properties such as something always being true or eventually becoming true. Its sheaf topology uses strictly interior subintervals, allowing local descriptions that need not specify behaviour at the endpoints. The resulting toposes are closely related, but not identical, and we shall have occasion to compare them.

There is another reason to take histories seriously. In the *behavioural approach* associated with Jan Willems, a system is characterized by the behaviours it allows, rather than by a particular choice of state equations [3]. Interconnecting systems then means requiring their histories to agree on shared variables. David Jaz Myers has emphasized how this perspective provides a common way of describing systems that may have quite different internal presentations; his categorical account, and joint work with Sophie Libkind, develop its compositional organization [4, 5]. This gives us an additional reason to study runners through their temporal executions. A runner tells us *how* an operation is implemented; a behaviour sheaf describes the possible histories of that implementation. How this passage relates to Garner's universal behaviour categories is a promising question for later, not something we shall assume at the outset.

These developments suggest a new setting in which to investigate Sweedler theory. We already know how runners describe the implementation of one theory of operations in another, possibly carrying internal state. What happens when the ambient category is itself a world of temporal behaviours? Can we turn a runner into a machine whose executions unfold over time? Does composition of runners correspond to interconnection of temporal machines? And can the temporal logic be used to express and verify guarantees about those implementations? We shall approach these questions through concrete examples, beginning with discrete observations and proceeding to continuous-time systems. The general question of when the full Sweedler construction exists in these toposes will remain in the background.

Relation to other parts. The runner equations come from Parts I–III; the finite-history and relative-observability questions from Parts V and IX; the potentials from Part VI; and the plant equations from Part X and XC. The witness carried across a monitor boundary recalls the arrow-topos calculation in Part VI.A. We use this interlude as a *bridge*, not as a replacement for any of those developments.

Relation to the two research papers. *Sweedler Runners for Temporal Systems, I: From Graded Operational Syntax to Temporal Behaviour* [6] begins with a chosen resource grading of Moore theory. It constructs observation-sorted polynomial trees, bounded comodels, a temporal-valued Chu pairing, and a comparison with completed Garner behaviours. Its input-word clocks are bounded branching tests, not yet physical interval trajectories. *Sweedler Runners for Temporal Systems, II: Resource-Certified Implementations and Ownership* [7] adds cost-writer residues, composable potential certificates, and witnessed ownership for restricted wiring. The present Interlude supplies the physical realisation: explicit runners become timed histories in the interval-sheaf settings of Schultz–Spivak–Vasilakopoulou and Schultz–Spivak, and their certificates yield concrete deadline guarantees. These three constructions are complementary; no general equivalence of their temporal categories or tensors is assumed.

2 External endofunctors and internal operations

We have seen throughout this tutorial that changing the ambient category changes the kinds of systems we can describe. In sets, a state is simply an element of a set. In the arrow topos, we could distinguish states or observations from witnesses certifying them. Now we want to work in a topos where objects describe behaviours extending over intervals of time. But how much of our theory of operations should change when we make such a move?

There are two possibilities worth distinguishing. We could retain familiar theories of operations, such as reading a value or updating a state, and investigate what their runners look like when interpreted as temporal machines. Alternatively, we could allow the operations themselves to be described within the temporal topos. A command might then involve a temporal signal, and the response it expects might itself be a behaviour over an interval. The second possibility is more ambitious: time would enter not only into the systems implementing a theory, but into the theory's operations.

This distinction has a categorical counterpart. Our usual starting point for Sweedler theory over a Grothendieck topos $\mathcal{E}$ is a suitable category of accessible endofunctors

$$[\mathcal{E}, \mathcal{E}]_{\mathrm{acc}},$$

or its $\mathcal{E}$-enriched, strong counterpart. These are *external* endofunctors: ordinary functors from $\mathcal{E}$ to itself. We can also describe operations using the internal language of $\mathcal{E}$. One particularly tractable class consists of *internal polynomial functors*, of the form

$$P(X) = \sum_{b:B} X^{E_b},$$

where B is an object of $\mathcal{E}$ and E_b is a family of objects indexed by B. We can read this as a collection of commands b, each with its own type E_b of possible responses. The important point is that these command and response types now belong to the ambient topos. Internal polynomials provide a structured class of operations, but they do not exhaust all external accessible endofunctors, nor all possible internal constructions.

We encountered a version of this distinction in the arrow topos $\mathcal{E} = \mathbf{Set}^{\to}$. Recall that its objects are functions $X_0 \to X_1$. For two such objects K and X, the internal exponential is

$$(X^K)_0 = \mathcal{E}(K, X), \qquad (X^K)_1 = \mathbf{Set}(K_1, X_1).$$

Thus an internal reader already takes account of the relationship between the two stages: at stage 0, its elements are commuting squares, not merely independent functions.

There are also accessible external endofunctors that cannot be expressed by internal polynomials. For example,

$$X \longmapsto (X_0 \xrightarrow{\mathrm{id}} X_0)$$

is an accessible endofunctor of the arrow topos. But the stage-1 component of an internal polynomial depends only on X_1, whereas this functor's stage-1 component is X_0. The distinction is therefore genuine, rather than simply two descriptions of the same operations.

For temporal sheaves, we may eventually want to investigate both approaches. In particular, allowing commands and responses to be temporal objects could lead to theories of interaction that are intrinsically sensitive to duration, causality, or synchronization. This raises substantial questions about arities and the existence of the corresponding Sweedler constructions, which we leave for a separate investigation.

Our approach in this Part will be more concrete. We shall begin with familiar operations and explicit stateful runners in **Set**, then ask how their executions can be organized into temporal behaviours. This will let us study the interaction between runners, time, and temporal verification without first having to settle the general existence problem.

3 Free observation theories and stateful runners

Let us return to one of our simplest examples: a program that can ask its environment for a value. The program issues a request, receives an answer, and chooses what to do next according to that answer. It may ask again, or it may eventually return a result. We have already seen how to describe such programs using a free monad, and it will be useful to recall the construction before introducing time.

For a set A of possible answers, let T_A be the free monad generated by a single observation operation with response set A. An element of $T_A(X)$ can be pictured as a well-founded tree. At each internal node, the program asks for an observation and branches according to the answer $a \in A$; at each leaf, it returns an element of X. We write

$$\mathsf{read}_A(k), \qquad k : A \to T_A(X),$$

for an observation followed by the continuation k, and $\eta(x)$ for a program that simply returns x.

For example, if $A = \{0, 1\}$, a program might read a Boolean value and, if the answer is 1, read another before returning. Nothing in the theory specifies where these answers come from. The theory describes what the program is allowed to ask and how its computations are assembled, but not the system that supplies the answers.

That is where runners enter. Suppose we have another theory R, describing the operations available in some environment. A runner explains how a program written using T_A can be executed using the operations of R. It may also keep some information between operations, such as the last value observed, a counter, or a record of earlier interactions. We call this additional information the *runner state*, or *residue*, and denote its set of possible values by S.

More precisely, a stateful runner is given by a state-passing monad map

$$T_A \longrightarrow \mathrm{St}_S(R), \qquad \mathrm{St}_S(R)(X) = \big(R(X \times S)\big)^S.$$

Equivalently, it is a natural family

$$\rho_X : T_A(X) \times S \longrightarrow R(X \times S) \tag{1}$$

respecting the monad unit and multiplication. Given a source program and an initial runner state, ρ_X produces a target computation. If that computation terminates, it returns both the source program's result and the updated runner state. The runner laws ensure that this translation respects the way programs are composed.

Fortunately, when the source theory is free, specifying a runner is much easier than defining the maps ρ_X separately for every X. We only have to say how the runner implements one observation. If the target is another free observation theory T_B, this amounts to choosing

$$\theta : S \longrightarrow T_B(A \times S). \tag{2}$$

For each current state s, the term $\theta(s)$ is a program using B-observations which eventually returns an A-value together with a new state. Whenever the source program asks for an A-observation,

the runner executes $\theta(s)$ and then continues the source program along the branch selected by the returned A-value.

To make this explicit, the runner acts on a returned value by

$$\rho_X(\eta(x), s) = \eta(x, s),$$

and on an observation node by

$$\rho_X(\mathsf{read}_A(k), s) = \theta(s)\mathsf{bind}\big((a, s') \mapsto \rho_X(k(a), s')\big),$$

where bind denotes substitution of a continuation into the leaves of a target computation. The universal property of the free monad ensures that this translation satisfies the runner laws.

As a small example, suppose $A = B = \{0, 1\}$ and the runner remembers the last Boolean value received. With $S = \{0, 1\}$, take

$$\theta(q) = \mathsf{read}_B\big(b \mapsto \eta(q, b)\big).$$

The runner asks the environment for a fresh bit b, returns the previously stored bit q to the source program, and stores b for next time. Thus a program sees each observation one step later than the environment supplied it.

This last example already hints at our temporal questions. An implementing computation may perform one target observation, several observations, or other operations before supplying the answer requested by the source program. Those operations take place in a particular order and, when interpreted by a physical system, may take time. We want to understand not just the final answer and updated state, but the *execution itself*: what happens along the way, how long it takes, and how successive executions fit together. This is what will lead us from runners to temporal machines.

For now, everything has been constructed in **Set**. We have used only the free-monad universal property, without assuming the existence of a measuring comonoid $\{T_A, T_B\}$. The more general Sweedler existence question remains separate from the concrete temporal interpretation we are about to develop.

3.1 A one-read implementation

There is a particularly simple case of the construction just described. Suppose that, whatever the current runner state, implementing one A-observation requires exactly one B-observation. The implementing computation $\theta(s)$ then reads a value $b \in B$ and immediately returns an A-value together with an updated state. All this information is captured by a function

$$f : S \times B \longrightarrow A \times S, \qquad f(s, b) = (a, s').$$

But this is just a *Mealy machine*: given its current state and an input, it produces an output and a new state. The only point to notice is the direction of the inputs and outputs. The physical environment supplies the B-observation, while the runner delivers the resulting A-observation to the source program.

Recall our delayed-read example, where $A = B = S = 2$ and

$$f(q, b) = (q, b).$$

It receives a fresh bit b, returns the previously stored bit q, and then stores b. If we number successive interactions by i, its behaviour is

$$q_{i+1} = b_i, \qquad a_i = q_i, \qquad \text{so} \qquad a_{i+1} = b_i.$$

We can now recognize this runner as a familiar state machine. Our next task is to see how such a machine can be described not just by its successive transitions, but by its behaviour over whole intervals of time.

4 One-read runners unfold into discrete interval sheaves

We have recognized our one-read runner as a Mealy machine. But a Mealy transition describes only *one* interaction. If a machine runs for ten steps, we want to talk about the whole ten-step history, and also about what happens when we look at just steps four through seven. This is where the interval-sheaf viewpoint becomes useful.

Think again of a machine with transition

$$f : S \times B \longrightarrow A \times S.$$

Starting in state s_0, it receives $b_0 \in B$, outputs $a_0 \in A$, and moves to s_1. It then receives b_1, outputs a_1, and so on. A history of length n is consequently a sequence

$$(s_0, \ldots, s_n;\ b_0, \ldots, b_{n-1};\ a_0, \ldots, a_{n-1}) \tag{3}$$

with

$$f(s_i, b_i) = (a_i, s_{i+1}) \qquad (0 \le i < n).$$

There is one more state than there are observations: the final state is the place where the next part of the execution may begin.

Now take a shorter interval of this history, beginning at step j. Its first state is s_j, *not* the original s_0. Conversely, two histories can be joined end to end when the final state of the first is the initial state of the second. These are precisely the restriction and gluing operations we want a temporal object to possess.

In the discrete interval-sheaf topos $\mathcal{E}_d$ of [2], sheaves of this kind can be described by directed graphs: vertices are states, edges are possible steps, and sections over an interval of length n are composable paths of n edges. For an alphabet B, the *signal sheaf* $\underline{B}$ has one vertex and one loop for each symbol. Thus

$$\underline{B}(0) = 1, \qquad \underline{B}(n) = B^n \quad (n \ge 1).$$

The distinction is worth noticing. A signal symbol labels an edge, representing what arrives during a step; it does not itself specify a state at the boundary of that step.

Let $\mathsf{Tr}(f)$ denote the sheaf of histories (3). Forgetting the states and outputs gives its input signal; forgetting the states and inputs gives its output signal. We therefore obtain a span

$$\boxed{\underline{B} \longleftarrow \mathsf{Tr}(f) \longrightarrow \underline{A}.}$$

This is the machine as it appears in the sheaf semantics. Unlike an input/output function, the middle object retains all the possible internal histories. That retention will be essential when machines are connected.

4.1 Connecting two histories

Suppose another runner supplies the B-observations required by f. Write its transition as

$$g : U \times C \longrightarrow B \times U.$$

At each step the second machine receives a physical C-value, produces a B-value, and updates its U-state. The first machine consumes that very B-value, produces an A-value, and updates its S-state. The combined machine therefore carries a pair (s, u) of states.

How should the history of the combined machine be described? We could calculate its transition function and unfold it. Or we could take one history of g and one history of f and require their intermediate B-signals to agree. The second description is a pullback of sheaves. The pleasing fact is that both descriptions coincide.

Proposition 4.1 (synchronous composition)**.** *Let $f : S \times B \to A \times S$ and $g : U \times C \to B \times U$ be deterministic Mealy transitions, and let h be their serial composite on state $S \times U$. There is a canonical isomorphism of discrete interval sheaves over the external ports*

$$\mathsf{Tr}(h) \cong \mathsf{Tr}(g) \times_{\underline{B}} \mathsf{Tr}(f). \tag{4}$$

The isomorphisms respect identity transitions and associative serial composition.

Proof. A length-n history on the right consists of two histories sharing the same B-word. At every step we have

$$g(u_i, c_i) = (b_i, u_{i+1}), \qquad f(s_i, b_i) = (a_i, s_{i+1}).$$

Pairing the states gives exactly a history of h. Conversely, a history of h uniquely determines the two histories and their shared word. The two procedures commute with taking subintervals. With three machines, either order of pairing gives the same compatible triple of histories; the stateless identity transition supplies the unit. □

This simple calculation is the first bridge we were looking for. *Composing implementations* and *wiring together temporal histories* agree, at least when one source read uses exactly one target read. Notice that the result identifies the sheaves of complete histories, not just the functions taking input words to output words. And we have proved it directly, without needing a measuring comonoid in the interval-sheaf topos.

Relation to bounded input charts. Paper I [6] organizes finite Moore observations over an input-word and budget category, keeping all possible future input words in one bounded behaviour. Each section of the present path sheaf instead records a *single linear execution* together with its boundary states, while the sheaf provides their restriction and adjacent-interval gluing. Fixing an input word in Paper I suggests such a chart, but obtaining the trajectory above also needs an explicit runner and its physical realisation. This distinction prevents us from identifying the graded comodel presheaf topos with the interval-sheaf topos.

5 Two reads per source operation: phase matters

The preceding machines all share a clock: every physical step produces one logical observation. A runner need not behave like that. It might have to ask its environment several questions before it can answer a single question from the source program.

Here is the smallest interesting example. With $A = B = 2$, consider the stateless runner

$$\theta(*) = \mathsf{read}_B\,(b_1 \mapsto \mathsf{read}_B(b_2 \mapsto \eta(b_1 \oplus b_2, *)))\,. \tag{5}$$

It reads two bits and returns their XOR. A physical stream $(c_0, c_1, c_2, c_3, \ldots)$ is seen by the source as

$$(c_0 \oplus c_1,\ c_2 \oplus c_3, \ldots).$$

For instance, the physical input $(1, 0, 1, 1)$ produces the two logical observations $(1, 0)$. There is a logical observation after the second physical tick, not after the first.

What happens if we stop the execution after only one physical tick? We cannot throw that history away: the machine has received b_1 and is waiting for b_2. To represent that partial execution, introduce two sorts of control configuration,

$$\mathsf{ready}, \qquad \mathsf{busy}(b_1),$$

with transitions

$$\mathsf{ready} \xrightarrow{b_1} \mathsf{busy}(b_1), \qquad \mathsf{busy}(b_1) \xrightarrow{b_2 \,/\, (b_1 \oplus b_2)} \mathsf{ready}.$$

Only the second arrow completes a source observation. The value b_1 is held only while the implementing computation is underway.

This brings out a distinction we shall need repeatedly. The runner's *residue* is memory that survives from one completed source operation to the next. Its *execution phase* records how far we have got through the implementation of the *current* operation. The XOR runner has no persistent residue, but it certainly has a nontrivial execution phase.

5.1 From implementing trees to event-clocked machines

The same idea works beyond this two-read example. Recall that $\theta(s) \in T_B(A \times S)$ is a tree describing how to implement one source read. While the tree is being evaluated, the machine must remember which subtree remains to be executed. At a leaf it obtains an A-response and a new runner state s'; it then becomes ready for the next source operation.

We concentrate on *positive well-founded* implementations. Every path from root to leaf performs at least one target observation, and there is no infinite evaluation path through an individual tree. The free-monad terms are well-founded already. Positivity ensures that a completed source operation does not happen without advancing the physical read clock. Neither assumption supplies a uniform numerical bound on the number of reads: different branches or different runner states can require arbitrarily many.

Why no uniform depth bound yet? That would rule out useful composites. For example, let an outer implementation perform two reads of a singleton response type $B = 1$. Let the inner runner have state $u \in \mathbb{N}$ and responses $C = \mathbb{N}$. At $u = 0$ it performs one read and updates u to $c + 1$ on response c. At $u = n \geq 1$ it performs n reads, then returns to $u = 0$. Every individual inner tree has finite height, but the composite beginning at $u = 0$ has branches of length $c + 2$, with no common bound as c varies. What survives composition is positivity and well-foundedness. We impose uniform estimates only when we begin studying durations in Section 6.

To keep physical time visible, add a special symbol $\perp$ to the logical event alphabet:

$$A_\perp = A + \{\perp\}.$$

An output $\perp$ means "one physical tick has passed, but no source observation has yet completed." The residual-term execution machine then has an ordinary Mealy transition

$$\widehat{f} : E_\theta \times B \longrightarrow A_\perp \times E_\theta.$$

A target read either advances the residual subtree and emits $\perp$, or reaches a leaf and emits its A-value. When another runner consumes these events, it simply waits through the $\perp$ ticks. This *stuttering extension* lets machines with different logical operation lengths share a physical clock.

5.2 Substitution still agrees with execution

We now have enough structure to revisit composition. Suppose an outer runner uses B-reads, while an inner runner realizes each such read by an entire C-computation. The composite program is formed by substituting an inner implementation tree at each outer B-read. During its evaluation we may be partway through the outer tree, and also partway through the current inner tree. Recording both residual computations is exactly what makes the temporal description work.

Proposition 5.1 (positive well-founded substitution)**.** *Let $\rho : T_A \rightsquigarrow T_B$ and $\gamma : T_B \rightsquigarrow T_C$ be stateful positive runners whose implementing trees are well-founded. A completed execution of $\gamma \odot \rho$ is canonically equivalent to a nested execution in which each B-read of ρ is implemented by a completed C-execution of γ. This correspondence preserves the physical C-word, the completed A-events, persistent states at the appropriate boundaries, and physical prefixes once residual control configurations are retained.*

Proof. Substitution tells us where every inner computation begins and where it returns. When the inner residual reaches a leaf, it supplies a B-value and updated inner state; the B-value selects the next branch of the outer residual. When the outer residual reaches a leaf, the A-operation is complete. Reading a substituted execution with these return points marked reconstructs the nested execution, and flattening the nesting gives the inverse construction. Prefixes agree because the residual configurations record unfinished work. Associativity follows from associativity of free-monad substitution; identity runners perform one read with no extra phase. □

Applying Proposition 4.1 to the stuttering event-clocked Mealy machines now gives pullback composition of their path sheaves. The direct residual-term machine of a syntactically substituted runner may not have literally the same state set as the serial event machine: one presentation may contain unreachable or differently nested control configurations. After the canonical encoding and, when needed, restriction to reachable or normalized configurations, their observed execution histories agree. This is the level at which we claim compositionality.

What we are leaving out. A zero-read implementation can complete a source operation without consuming a physical tick. A possibly divergent target computation need never complete one at all. Neither situation can be represented faithfully by simply inserting an ordinary $\perp$ tick. Our current construction is deliberately about positive, terminating implementations; silent completions and divergence require a richer timed-event semantics.

6 Time distortion and amortised certificates

We have learned how to represent the time *within* an implementing computation. The next question is quantitative: how much physical time can a logical computation use?

At present one physical read takes one tick. If every source operation uses between one and m target reads, and τ_i is the physical tick at which the ith logical operation completes, then

$$j \leq \tau_{i+j} - \tau_i \leq mj.$$

And if one outer operation invokes at most m inner reads, each costing at most n physical reads, its worst-case cost is at most mn. This is reassuringly simple. It is also often a poor estimate.

Imagine a runner which alternates between expensive and cheap operations: three physical reads, then one, then three, then one. Calling every operation a three-tick operation forgets the

alternation. We need a way of recording how the runner's current state makes future work more or less expensive. This is the role of a *potential*, just as in the amortised resource calculations of Part VI.

Definition 6.1 (duration certificate)**.** For a positive well-founded runner with persistent state S, an *amortised duration certificate* consists of a rate $r \geq 0$ and a potential $\Phi : S \to \mathbb{R}_{\geq 0}$ such that every completed transition $s \xrightarrow[d]{a} s'$ obeys

$$d + \Phi(s') \leq r + \Phi(s). \tag{6}$$

Read the inequality as a budget rule. We are allowed r units of cost for this operation, plus any credit $\Phi(s)$ available at its start. The operation spends d units and must leave at least $\Phi(s')$ units of credit for the future. An expensive step is permissible if it reduces the stored credit; a cheap one can rebuild it.

For consecutive operations, these inequalities add, and every intermediate potential occurs once on each side. The result is

$$D_k + \Phi(s_k) \leq rk + \Phi(s_0). \tag{7}$$

If $\Phi \leq K$ on reachable states, this gives $D_k \leq rk + K$. Instead of charging the largest possible cost m on *every* operation, we pay a rate r plus a bounded initial allowance.

6.1 A small example before the composition theorem

Take the alternating three-tick/one-tick runner. Let its phases be slow and fast, and choose

$$r = 2, \qquad \Phi(\text{slow}) = 1, \qquad \Phi(\text{fast}) = 0.$$

The slow transition uses up its one unit of credit: $3 + 0 = 2 + 1$. The fast transition restores the credit: $1 + 1 = 2 + 0$. So both satisfy (6) with equality, and a run starting in the slow phase obeys

$$D_k \leq 2k + 1$$

instead of the crude estimate $3k$. The estimate expresses something we can see directly in the alternating schedule: two successive operations cost four ticks, not six.

6.2 Why certificates compose

It is not enough to have a good estimate for one runner. The point of Sweedler delegation is that an implementation can itself be implemented, possibly by a further runner. We therefore want to reuse a timing certificate when we substitute one runner into another.

Theorem 6.2 (composition of duration certificates)**.** *Suppose $\rho : T_A \rightsquigarrow T_B$ has certificate (r, Φ), measuring cost in completed B-operations, and $\gamma : T_B \rightsquigarrow T_C$ has certificate (q, Ψ), measuring physical C-operations. Then the composite has rate rq and potential*

$$\Theta(s, u) = q\Phi(s) + \Psi(u). \tag{8}$$

In particular, for its completed A-operation of physical duration D,

$$D + \Theta(s', u') \leq rq + \Theta(s, u).$$

Proof. Suppose one outer operation makes j inner calls. Its certificate says $j + \Phi(s') \le r + \Phi(s)$. If the inner calls have costs $d_1, \dots, d_j$, their inequalities add to

$$D + \Psi(u') \le qj + \Psi(u), \qquad D = \sum_{i=1}^{j} d_i.$$

Multiply the outer inequality by $q \ge 0$ and add. The intermediate number qj cancels, giving the claimed composite potential. □

Why does the outer potential become $q\Phi$? Because the outer runner measures its saved work in *inner operations*, and each such unit is budgeted at q physical steps. The rescaling therefore has a simple operational meaning. It is also associative:

$$p(q\Phi + \Psi) + \Xi = pq\Phi + p\Psi + \Xi,$$

with unit certificate $(1, 0)$. We have a compositional resource law for this class of runners, without claiming a corresponding theorem for arbitrary ordered cost monoids.

Relation to cost-writer certified runners. The same potential-composition formula reappears in Paper II [7]: a second-stage writer implementation with cost translation f changes the composite potential to $f(\Phi) + \Psi$. For additive numerical rates $f(c) = qc$, this gives exactly $q\Phi + \Psi$ above. Paper II retains actual cost and allowances as writer-residual data and adds ownership-sensitive framing; here we instead attach elapsed duration to completed physical executions. The logical-to-physical deadline theorem below uses this extra interpretation of time and is not a consequence of a writer quotient alone.

7 Transporting a bounded-response specification

What can a duration certificate tell us about a program's correctness? Consider the ordinary delayed-read runner from Section 3. If an input bit is 1, the runner delivers that bit at the *next* completed observation. We might interpret a 1 as a request that is acknowledged one logical step later. But if each logical step takes a variable number of physical ticks, "next step" does not yet tell us how long the acknowledgement will take.

Let P_i mean that a request occurs at the ith *completed logical event*, and Q_i that the same event carries an acknowledgement. Suppose the source-level specification is

$$\forall i \; (P_i \Rightarrow \exists j \in \{1, \dots, k\} \; Q_{i+j}). \tag{9}$$

It promises an acknowledgement at one of the next k logical events. Write τ_i for the physical time when event i completes. The question is how far apart τ_i and τ_{i+j} can be.

One qualification is essential. A *physical* observation need not be the same event as a *logical* observation. The XOR runner, for example, combines two physical bits into one logical bit. Throughout this theorem, P and Q refer to the logical events returned by the runner and then marked at their physical completion times, not to arbitrary raw physical inputs.

Theorem 7.1 (bounded-response refinement)**.** *Assume that executions are total and time-divergent, that* (9) *holds, and that the runner has certificate* (r, Φ) *with* $0 \le \Phi \le K$ *on reachable states. Then every physical request event has an acknowledging completion event within* $rk + K$ *physical ticks (rounding up gives an integer deadline).*

Proof. Take a request at logical event i. By the source specification, it has an acknowledgement at event $i+j$ for some $1 \le j \le k$. Apply the telescoping estimate (7) to those j operations:

$$\tau_{i+j} - \tau_i \le rj + \Phi(s_i) - \Phi(s_{i+j}) \le rk + K.$$

Thus the acknowledgement occurs within the stated physical deadline. □

For delayed reading, $k = 1$. Feed it through our alternating three-tick/one-tick implementation, for which $r = 2$ and $K = 1$. The theorem gives a deadline of three physical ticks. A request completed just before a slow phase really does have to wait all three ticks for the following logical event, so the bound is sharp.

We have moved from a qualitative statement about the *next* logical observation to a quantitative statement about elapsed physical time. The duration certificate is what makes the translation reliable. The next step is to make a verifier that can check the resulting physical deadline as the machine runs.

8 Timed monitors: turning deadlines into state safety

An acknowledgement deadline is not a property of one isolated event. When a request appears, it creates an obligation that remains in force as time advances. A verifier must remember that obligation until an acknowledgement arrives or the deadline expires.

This helps explain a subtle point about temporal sheaves. Suppose the deadline is one tick. The single-event word P is not yet a violation: there has been no following tick on which an acknowledgement could arrive. The single-event word $\varnothing$ is harmless too. But putting them together produces $P\varnothing$, which misses the deadline. So the collection of acceptable *unannotated* finite words need not satisfy the sheaf gluing axiom. This is why [2, §4.5] treats observable safety contracts as subpresheaves.

The remedy is to keep track of the pending obligation. Fix an integer deadline $N \ge 1$ and an event alphabet D equipped with request and acknowledgement predicates P, Q. A simple monitor has states

$$\mathsf{idle}, \quad \mathsf{bad}, \quad 0, 1, \dots, N-1.$$

The numbered states record the age of the *oldest* request that has not yet been acknowledged. We need only the oldest because a single Q-event discharges every previously pending request.

Here is the timing convention. At a tick carrying Q, first discharge any older requests; if that same tick also carries a new P, register it as a fresh obligation of age zero. Without acknowledgement, the oldest obligation ages by one tick, and the monitor enters bad as soon as its last permitted response opportunity is missed. The state bad is absorbing. For infinite streams beginning with an idle monitor this gives exactly

$$\mathsf{bad} \text{ is never entered} \quad \Longleftrightarrow \quad \forall i \, (P_i \Rightarrow \exists j \in \{1, \dots, N\} \, Q_{i+j}).$$

Thus a bounded-response property has become an ordinary state-safety condition: never reach bad.

8.1 The monitor as another temporal machine

Make the monitor into a directed graph G_N: its vertices are monitor states, and an edge labelled $d \in D$ records the corresponding state transition. Its path sheaf Mon_N comes with an event map $\mathsf{Mon}_N \to \underline{D}$.

Now let a machine have execution sheaf X and event map $e : X \to \underline{D}$. To run the monitor alongside the machine, we pair histories with the same observed event word. Categorically this is the pullback

$$X_N = X \times_{\underline{D}} \mathsf{Mon}_N. \tag{10}$$

A section of X_N remembers both the machine execution and the monitor's evolving state. Removing the vertex bad and its incident edges gives the good-monitor subgraph, and hence a subobject

$$X_N^{\mathrm{good}} \hookrightarrow X_N.$$

Unlike the unannotated accepted-word collection, this is a sheaf: when two monitored histories meet, their monitor states at the shared boundary must agree. A pending request cannot silently disappear between intervals.

The use of a pullback also tells us how monitoring interacts with machine interconnection. It should not matter whether we first wire two machines together and then monitor their shared result, or attach the monitor to the appropriate output and then perform the wiring.

Proposition 8.1 (monitoring commutes with wiring)**.** *For serially composable machine spans $\underline{C} \leftarrow Y \to \underline{B}$ and $\underline{B} \leftarrow X \to \underline{A}$, and any monitor $M \to \underline{A}$, there is a canonical isomorphism*

$$(Y \times_{\underline{B}} X) \times_{\underline{A}} M \cong Y \times_{\underline{B}} (X \times_{\underline{A}} M).$$

Proof. Both sides describe the same triples of histories: one from Y, one from X, and one from M. The two machine histories have matching intermediate B-signals, and the monitor sees the external A-signal. Rearranging the brackets does not change these conditions or their compatibility with restriction. This is the pullback pasting law. □

The proposition says that the *construction of the monitored history object* is compositional. It does not assert that an arbitrary machine meets the deadline. To prove that, we combine a source-level guarantee with a duration certificate, as in Theorem 7.1, and conclude that an appropriately initialized reachable execution stays inside X_N^{good}.

Where the witnesses matter. A monitored path records the age of the outstanding request at every interval boundary. Forgetting that age may destroy the possibility of gluing accepted histories. This is reminiscent of the witness-based arrow-topos examples of Part VI.A: the truth-valued shadow alone can hide the information needed to make two pieces compatible. The categorical image in a sheaf topos is therefore not automatically the sectionwise image of all acceptable finite histories.

9 A genuinely continuous-time runner

So far, our physical clock ticks whenever the target environment supplies a symbol. Real systems need not behave this way. A sensor may be read only after a delay, and the duration of that delay may depend on what the controller observed earlier. We would like one source operation to run a small *physical procedure*: wait, sense, adjust the actuator, return.

Consider the plant

$$\dot{x}(t) = -\lambda x(t) + v(t), \qquad \lambda > 0, \tag{11}$$

where x is the measured state and v is an actuator setting, held constant between sampling events. We give the physical environment a free theory with commands wait_d for $d \geq 0$, sense with a real-valued response, and set_w for $w \in \mathbb{R}$.

Those are still just *commands*. The physical plant tells us what they do. In plant state (x, v), executing them gives

$$\begin{aligned} \mathsf{wait}_d(x,v) &= \left(e^{-\lambda d}x + \frac{1-e^{-\lambda d}}{\lambda}v,\ v \right), \\ \mathsf{sense}(x,v) &= (x,(x,v)), \\ \mathsf{set}_w(x,v) &= (x,w). \end{aligned}$$

For instance, sense returns the present value of x without changing the physical state, while wait_d advances the solution of the differential equation for d seconds. We could later impose the equation $\mathsf{wait}_d; \mathsf{wait}_e = \mathsf{wait}_{d+e}$, which the plant satisfies. Beginning with the free theory lets us describe the runner without first working through that quotient.

9.1 A sampler that chooses its next delay

Let the runner remember the last measurement s. Choose the positive waiting time

$$h(s) = 1 + \frac{s^2}{1+s^2}, \qquad 1 \leq h(s) < 2,$$

and let $K : \mathbb{R} \to \mathbb{R}$ be a continuous control policy. The generic source observation is implemented by the following program:

$$\mathsf{wait}_{h(s)}; \quad b \leftarrow \mathsf{sense}; \quad \mathsf{set}_{K(b)}; \quad \mathsf{return}(b,b). \tag{12}$$

Read it in order: wait for a duration chosen from the stored sample, read the plant, set the new control, and return the fresh reading both to the source program and as the runner's updated memory. Because the source theory is free, this implementing program determines a stateful runner in **Set**.

Suppose a sampling event has just completed, with stored sample $s = x$ and plant state $z = (x, v)$. The next measurement is

$$x^+ = e^{-\lambda h(x)}x + \frac{1-e^{-\lambda h(x)}}{\lambda}v, \qquad F(x,v) = (x^+, K(x^+)). \tag{13}$$

Thus the completed operations define a discrete return map F, but with a *state-dependent elapsed time* $h(x)$ attached to each transition. Our old Mealy picture has gained a genuine duration.

9.2 Continuous suspension and its interval sheaf

How do we turn this event-to-event description into a continuous-time behaviour? Between one sample and the next, the plant continues evolving. We must remember not only the state at the last sample but also how far into the current waiting period we have travelled.

Given a continuous return map $F : Z \to Z$ and a continuous waiting-time function $h : Z \to (0, \infty)$ bounded below by $\varepsilon > 0$, define

$$M_h = \{(z,u) : 0 \leq u \leq h(z)\} / \big((z, h(z)) \sim (Fz, 0)\big). \tag{14}$$

Here z is the last post-sampling state and u the time since that sample. We increase u at unit speed. When u reaches the chosen roof $h(z)$, we identify the end of this waiting period with the start of the next, in state $F(z)$. This construction is called a *suspension*. The positive lower bound on h prevents infinitely many jumps from being squeezed into finite time, and we obtain a forward semiflow φ_t.

A section over $[0, \ell]$ is now an entire piece of this semiflow:

$$H_h(\ell) = \{t \mapsto \varphi_t(m) : m \in M_h,\ 0 \le t \le \ell\} \cong M_h.$$

If we restrict to a subinterval beginning r seconds later, its initial state is $\varphi_r(m)$. Two adjacent pieces glue exactly when the end of the first is the beginning of the second. The semiflow law makes that gluing unique. This is the continuous analogue of the finite paths of Section 4, now with unequal, state-dependent waiting periods.

The physical variable x is continuous across a sampling event, because the controller changes v, not x. This does not depend on continuity of K; rather, continuity of K is used in establishing the continuous return map and suspension. We may therefore extract a continuous x-signal from the execution sheaf, while the jumping actuator v belongs to a hybrid or piecewise-continuous signal type. The logical value returned by the runner must be retained in an event channel: it cannot in general be reconstructed from the physical x-signal alone.

9.3 The delayed reading acquires a real deadline

Compose the sampler with the familiar delayed-read runner, now with $A = B = S = \mathbb{R}$ and $f(q, b) = (q, b)$. At sampling event i the sampler supplies a fresh measurement b_i and the delayed runner returns a_i. We still have $a_{i+1} = b_i$, but the elapsed time between these two events is now variable.

Proposition 9.1 (real-time delayed response)**.** *For the composite described above, if τ_i are successive sampling times, then*

$$1 \le \tau_{i+1} - \tau_i < 2.$$

For any predicate P on sampled measurements, every complete execution satisfies: a completion event with $P(b_i)$ is followed by a completion event with $P(a_{i+1})$ less than two seconds later. The execution is non-Zeno.

Proof. Delayed reading gives $a_{i+1} = b_i$. The sampler schedules the next event after $h(b_i)$ seconds, which lies in $[1, 2)$. Finally, since every waiting period lasts at least one second, the event times tend to infinity. □

This is the first example in which the runner has been given a genuinely continuous physical execution, and a temporal guarantee follows from its implementation. The simple logical statement "the next observation returns the previous measurement" has become "the measurement is returned within two seconds."

There is an instructive variation. We could impose a timeout at $D = 1.5$ seconds, returning either $\mathsf{sample}(b)$ or $\mathsf{timeout}$. Then the response type must change from $\mathbb{R}$ to $\mathbb{R} + 1$: a failed observation is not silently converted into a successful one. A timeout may require no sensing operation, although it consumes positive elapsed time. Thus the continuous-time analogue of our positivity assumption ought ultimately to concern *time elapsed*, not merely the number of target reads.

10 Real-duration amortisation and non-Zeno obligations

Our discrete potential argument carries over to elapsed seconds. The only change is in the units. If an outer runner budgets a number of inner operations and each inner operation takes physical time, the same telescoping calculation as in Theorem 6.2 gives

$$D + \big(q\Phi(s') + \Psi(u')\big) \le rq + q\Phi(s) + \Psi(u).$$

Here r is measured in inner operations per outer operation, q in seconds per inner operation, and rq in seconds per outer operation. The factor $q\Phi$ has precisely the right units for the composite potential.

For a numerical example, suppose a sampler alternates between a slow wait of 1.8 seconds and a fast wait of 0.8 seconds. The worst single operation lasts 1.8 seconds, but charging that much for every operation forgets that the phases alternate. Choose

$$r = 1.3, \qquad \Phi(\text{slow}) = 0.5, \qquad \Phi(\text{fast}) = 0.$$

The two transitions satisfy

$$1.8 + 0 = 1.3 + 0.5, \qquad 0.8 + 0.5 = 1.3 + 0.$$

Hence a run of k operations starting in the slow phase obeys

$$D_k \le 1.3k + 0.5,$$

in place of the crude $1.8k$. We will reuse this very sampler as our simple oscillator when we compare the two temporal toposes.

10.1 An upper bound is not a lower bound

There is one phenomenon that did not arise with discrete, positive-read runners. Suppose successive operations take $\frac{1}{2}, \frac{1}{4}, \frac{1}{8}, \dots$ seconds. Each duration is positive, but infinitely many operations finish within one second. Such an execution is called *Zeno*. A good upper bound on response time does nothing by itself to prevent it.

A sufficient non-Zeno condition is a uniform lower dwell time $h \ge \varepsilon > 0$. More generally, we can require that the sum of durations diverges along every infinite execution. For nested runners, a uniform lower bound is inherited when every outer operation requires at least one inner operation of duration at least ε. Keeping *upper* timing guarantees separate from *time divergence* will matter in any future extension to general hybrid or nondeterministic systems.

11 Two interval toposes, not one

We have now built runner executions as sheaves of histories, first in discrete time and then in continuous time. We can at last return to the distinction announced in the introduction. Why did Schultz, Spivak, and Vasilakopoulou use one interval-sheaf framework for machines, while Schultz and Spivak used a different one for temporal logic?

The issue is what we expect to know at the *boundary* of an observed interval. For machines it is natural to retain the state at both ends: the final state of one execution is the initial state of the next. For temporal reasoning we may instead want to say that a behaviour is determined by what

happens throughout the *interior* of an interval, without insisting that it have a specified endpoint value. Both choices can be expressed by sheaf conditions, but they are different conditions.

Let $\mathsf{Int}_{\mathbb{R}}$ be the category whose objects are lengths $\ell \geq 0$. A morphism selecting a subinterval of length ℓ' is described by two margins $r, s \geq 0$, with

$$\ell = r + \ell' + s.$$

The topology J used for interval machines in [2] is generated by covers that split an interval into adjacent closed pieces. Write

$$\mathcal{E}_J = \mathrm{Sh}(\mathsf{Int}_{\mathbb{R}}, J).$$

In this world, compatible histories agree at the shared boundary and then glue to a longer history. Our suspension sheaf H_h belongs here.

The temporal topos of [1] uses instead a coverage K consisting, for an interval of length ℓ, of *all strictly interior* subintervals, those for which $r, s > 0$. Following their site and opposite-site conventions, write

$$\mathcal{B} = \mathrm{Sh}(\mathsf{Int}_{\mathbb{R}}, K).$$

The duration-zero object has an empty covering family. Its sheaf condition consequently forces

$$X(0) \cong 1 \qquad (X \in \mathcal{B}). \tag{15}$$

That may look startling: where have the states of a machine gone? We shall calculate the answer for our oscillator in the next section. The important point for now is that (15) says nothing by itself about how many *positive-duration* behaviours there are.

11.1 Why the distinction is real

Here is a small test that prevents us from treating the two categories as the same. Choose a set D with at least two elements. Put D over every positive interval, with identity restrictions between positive intervals, and put a singleton over zero. This is a K-sheaf: all strictly interior pieces of a connected interval must carry the same label. But it is not a J-sheaf. Two adjacent positive intervals can carry different labels, agree automatically at their zero-length common boundary, and nevertheless have no common label on their union.

Thus $\mathcal{B}$ is *not included in* $\mathcal{E}_J$ through the evident inclusion into presheaves on the common interval category. This statement does not exclude a different geometric embedding of the toposes.[1]

Even without such an inclusion, there is a straightforward way to compare a machine sheaf with a temporal-logic sheaf. First forget that it satisfies the J-sheaf condition, then sheafify for K:

$$L : \mathcal{E}_J \longrightarrow \mathcal{B}, \qquad L = a_K \circ U_J. \tag{16}$$

The functor U_J includes J-sheaves among presheaves, and a_K is K-sheafification. Each preserves finite limits, so

$$L(X \times_Z Y) \cong LX \times_{LZ} LY. \tag{17}$$

This is the fact we shall use later: wiring and monitor constructions made from pullbacks survive this comparison. It does *not* make L a reflector into an evident subcategory of $\mathcal{E}_J$, nor does it promise to preserve all temporal logical connectives.

[1] The introductory discussion in [1, §1.4] describes $\mathcal{B}$ as a subtopos obtained by imposing continuity gluing. If its ambient $\mathrm{Shv}(\mathsf{Int})$ is read as precisely $\mathcal{E}_J$ inside the same presheaf category, that description is in tension with the example above and [1, Def. 3.5]. The example establishes non-inclusion through the evident presheaf embedding; it does not settle every possible comparison between the toposes.

Relation to the topos of bounded futures. The graded comodels of Paper I [6] live over the category of input-labelled numerical budgets; for example their terminal bounded future has a genuine observation set at grade zero. Neither interval-sheaf topos above is identified with that presheaf topos. In particular, the strict-interior topology forces a singleton section at duration zero, while positive-duration sections can retain distinct phases. Comparing budgeted comodel evaluation with either interval-sheaf realisation therefore requires additional structure, not just a relabelling of grades.

12 Exact sheafification of the oscillator

We could try to calculate L for the adaptive differential-equation controller. But the alternating-duration sampler from Section 10 is a much better place to begin. It has genuine unequal real durations, yet its complete behaviour is extremely simple.

Let a phase-0 interval last 1.8 seconds and a phase-1 interval last 0.8 seconds; at each completion the phase toggles. After both phases we return to where we started, so the full cycle lasts

$$1.8 + 0.8 = 2.6 \text{ seconds.}$$

The suspension of this system is a circle

$$C = \mathbb{R}/(2.6\mathbb{Z}),$$

whose phase evolves as $z \mapsto z + t$ modulo 2.6. Choose phase 0 for the start of the slow interval and phase 1.8 for the start of the fast one.

Let $H \in \mathcal{E}_J$ be its closed-interval trajectory sheaf. A history of any length ℓ is determined by its starting phase $z \in C$:

$$H(\ell) = C.$$

If we begin a subinterval r seconds later, the new initial phase is $z + r$ modulo 2.6. In particular, $H(0) = C$: a zero-duration section remembers the phase of the machine at that boundary.

What does L do? Strictly interior observations no longer see the zero-duration boundary as a separate object. But they still distinguish where we are on the circle: knowing even a short trajectory and how translation acts lets us recover its phase.

Proposition 12.1 (oscillator sheafification)**.** *The temporal sheaf LH is described by*

$$(LH)(\ell) \cong \begin{cases} C, & \ell > 0, \\ 1, & \ell = 0, \end{cases} \tag{18}$$

with the original phase-translation restrictions between positive intervals and the unique restrictions to zero.

Proof. Call the displayed presheaf $\widehat{H}$. A compatible collection of short, strictly interior trajectories gives local phases on overlapping intervals. Since translation on C is invertible, each local phase can be transported back to a common reference time. Agreement on overlaps makes the resulting phase independent of the choice of local piece. The connected strict-interior cover therefore determines a unique global positive-duration section. At duration zero, the empty cover requires a single section. Thus $\widehat{H}$ is a K-sheaf. The map $H \to \widehat{H}$ is the identity at positive durations and the unique map at zero; it becomes an isomorphism after K-sheafification. □

So the phase has not been erased. It survives in every positive-duration observation, while evaluation at duration zero has become trivial. This is a useful example of what we might call a *time-relative state*: it is encoded by a local behaviour and its restrictions, rather than by a standalone zero-duration element. For more general systems, different internal states can still have the same observable behaviour, so recovering a phase in this example is not a theorem about all runner residues.

12.1 What happens to punctual events?

There is a second worry. Our runners return values at particular sampling instants. Does the temporal topology, by ignoring the endpoints of an interval, erase these punctual events?

Let $E_{\mathrm{cl}}(\ell)$ be the finite D-labelled event sets in the closed interval $[0, \ell]$, restricted by intersection. These form a J-sheaf. A strictly interior observation cannot distinguish two closed event histories that differ only at the two endpoints. Carrying out the K-sheafification gives

$$LE_{\mathrm{cl}}(\ell) \cong \begin{cases} \{\text{locally finite } D\text{-labelled event sets in } (0,\ell)\}, & \ell > 0, \\ 1, & \ell = 0. \end{cases} \tag{19}$$

The reason is instructive. An event set on the open interior is finite on every compact interior subinterval, so each of its local pieces can be represented by a finite closed-interval event history. Those compatible pieces glue. We need not require the events to have an endpoint limit: they may even accumulate toward an excluded endpoint.

Now imagine an event at absolute time t. It is absent from the open-interior event set when t is precisely the endpoint of the chosen window. But enlarge the window slightly so that it contains t in its interior: the event is visible there. *An event at a boundary has not disappeared from temporal behaviour; its description has moved to surrounding positive-duration intervals.*

Sheafification may do more than discard endpoint values. For the ODE $\dot{x} = -x^2$ on nonnegative states, the solution $x(t) = 1/t$ is defined on $(0, \ell)$ but has no finite initial state at $t = 0$. On every compact interval strictly inside $(0, \ell)$ it is a perfectly good solution, and those local pieces agree. The K-sheafification therefore admits a positive-duration history that is not a closed-interval history starting from a finite state. Our circle example avoids this: translation is invertible, so its local histories extend uniquely to the endpoints. For an adaptive hybrid plant, extension requires its own hypotheses and boundary conventions.

13 The clock object and the sharp bounded-response calculation

There is one more step we can take with the oscillator. We know that a request at phase 0 is followed by an acknowledgement at phase 1.8. We have also seen that the temporal topos has only one section at duration zero. So how can its internal logic talk about something that happens *at a particular time*?

The answer in *Temporal Type Theory* is to introduce a clock. Instead of asking for an instantaneous state as a section over a zero-length interval, we consider a behaviour *together with a chosen temporal origin*. This is the role of the object Time. For positive durations its sections are pairs specifying the clock readings at the two ends of the interval:

$$\mathsf{Time}(\ell) = \begin{cases} \{(d,u) \in \mathbb{R}^2 : u - d = \ell\}, & \ell > 0, \\ 1, & \ell = 0. \end{cases} \tag{20}$$

If a subinterval starts r seconds later and ends s seconds before the old endpoint, its clock pair changes from (d, u) to $(d + r, u - s)$. A restriction to duration zero is unique. We follow the positive-duration site discussed in [1, Rem. 3.6].[2] The close connection between clocks and interval semantics is made precise by [1, Prop. 3.11]:

$$\mathrm{Sh}(\mathsf{IR}) \simeq \mathcal{B}/\mathsf{Time}.$$

The clock reading 0 marks a time, even if that marked time lies outside the particular observation window.

13.1 Marking a request without a zero-duration state

A positive-duration section of $H \times \mathsf{Time}$ consists of an initial oscillator phase $z \in C$ and an initial clock reading $d \in \mathbb{R}$. Both increase by the same amount when we restrict to a later part of the interval. Their difference is therefore unchanged:

$$\alpha(z, d) = z - d \pmod{2.6}. \tag{21}$$

Indeed $(z + r) - (d + r) = z - d$. This defines a morphism $\alpha : H \times \mathsf{Time} \to \underline{C}$, where $\underline{C}$ denotes the constant-object sheaf associated with C. Intuitively, α is the oscillator phase at the time where the chosen clock reads zero. We can calculate it even when that time is not an endpoint of the current interval, because our oscillator flow is invertible.

Recall that we labelled phase 0 as a request event P, and phase 1.8 as an acknowledgement event Q. Thus the *marked-request* subobject is

$$\mathsf{P} = \alpha^{-1}(\{0\}) \hookrightarrow H \times \mathsf{Time}.$$

Its elements are oscillator behaviours accompanied by a clock marking a request. Notice that this is a perfectly meaningful nontrivial subobject, even though $H(0)$ and $\mathsf{Time}(0)$ are singletons.

13.2 Looking for a response witness

Fix a positive deadline D. Starting at a marked request, ask for a positive delay $\delta \le D$ after which the oscillator reaches acknowledgement phase 1.8. Introduce a witness object W_D over P whose additional datum δ satisfies

$$0 < \delta \le D, \qquad \alpha + \delta = 1.8 \pmod{2.6}. \tag{22}$$

Here δ belongs to the *constant real-number object*. The image of the projection $W_D \to \mathsf{P}$ is an internal subobject: it selects the marked requests for which such a response witness exists. The construction uses an ordinary internal existential, implemented categorically by taking the image of a projection.

This particular system makes the calculation almost transparent. At a marked request, $\alpha = 0$, so the positive possible response delays are exactly

$$1.8, \quad 1.8 + 2.6 = 4.4, \quad 1.8 + 2(2.6) = 7.0, \quad \ldots.$$

There is no acknowledgement earlier than 1.8 seconds, and one occurs exactly then. The internal calculation gives the same sharp threshold.

[2] In arXiv v3 of [1], Eq. (3.2) prints the pair formula without an explicit $\ell > 0$ qualification (and the formula reappears as Eq. (6.2)). Taken literally at $\ell = 0$, it would conflict with the empty-cover conclusion of Rem. 3.6. Here we use the equivalent positive-duration site and its prescribed singleton zero component.

Theorem 13.1 (sharp marked-event response)**.** *For the alternating oscillator, the image subobject is*

$$\mathrm{im}(W_D \to \mathsf{P}) = \begin{cases} 0_{\mathsf{P}}, & 0 < D < 1.8, \\ \mathsf{P}, & D \geq 1.8. \end{cases} \tag{23}$$

For $1.8 \leq D < 4.4$ *the witness projection is an isomorphism. For* $D \geq 1.8$ *it has a section selecting* $\delta = 1.8$*.*

Proof. On P we have $\alpha = 0$. The positive solutions of (22) are precisely

$$\delta = 1.8 + 2.6n, \qquad n = 0, 1, 2, \ldots.$$

There are none for $D < 1.8$. For $D \geq 1.8$ the constant choice $\delta = 1.8$ is an internal witness for every marked request. It is the unique choice while $D < 4.4$. These facts give the displayed images in the slice over P. □

13.3 Comparing the answer with a timed monitor

Imagine also attaching a real-time monitor to the marked request. It starts with a pending obligation of age 0 and increases that age as time advances. It accepts an acknowledgement at age r when $0 < r \leq D$, and rejects once the deadline is passed without one. For our oscillator, the next Q arrives at exactly $r = 1.8$. So the monitor accepts precisely when $D \geq 1.8$.

For this deterministic complete-execution example, define the monitor-acceptance subobject $\mathsf{MonSafe}_D \hookrightarrow \mathsf{P}$ by those calculated cases: it is 0_{P} for $D < 1.8$ and P for $D \geq 1.8$. Comparing with Theorem 13.1, we obtain

$$\boxed{\mathsf{MonSafe}_D = \mathrm{im}(W_D \to \mathsf{P}).}$$

The agreement is exact and the deadline is sharp. It is also important to say what kind of agreement this is: we have *calculated* monitor acceptance externally for this oscillator and identified its resulting subobject with the internally witnessed one. We have not independently defined the monitor using all the intrinsic clock modalities of $\mathcal{B}$ and proved a general modal-monitor equivalence.

Nor are finite fragments the same as complete executions. A short fragment may stop with a request pending and no response yet in sight. The monitor remembers the obligation, whereas a future witness may lie outside the fragment. Our equality concerns complete oscillator executions marked at a request, not arbitrary finite-prefix acceptance conditions.

What we have actually gained. Although a behaviour type in $\mathcal{B}$ has a singleton at duration zero, the pair $H \times \mathsf{Time}$ can specify a request at a chosen instant. An internal existential over a *constant elapsed duration* then detects the correct 1.8-second response threshold. This is a genuine internal subobject calculation, but it is not yet an identification with the full metric-temporal logic interpretation in [1, §8.6].

13.4 Towards a genuinely clock-indexed response proposition

The limitation points naturally to the next question. So far we introduced a constant real number δ and added it to the marked clock time. A more intrinsic temporal formula would quantify over a *second clock position*, expressing something like

$$\exists(t' : \mathsf{Time})\ (t < t' \leq t + D\ \wedge\ Q(t')). \tag{24}$$

Here t is the marked request time, and $Q(t')$ must mean an acknowledgement by the *same execution* at the later clock time t'. To make this precise requires the order on Time, the Dedekind real deadline, and reindexing operations that let us inspect the same behaviour at two marked times. Writing the formula is not yet a construction of all those maps.

For the oscillator, the expected witness is $t' = t + 1.8$. A general comparison between the internal predicate (24) and an initialized timed-monitor object remains open. It also brings us close to our dependent Sweedler programme: a runner's implementation and residue might be indexed over a chosen clock or temporal world, rather than interpreted against time entirely from outside.

14 Composition with monitors after temporal sheafification

We can finish the categorical story by returning to the pullbacks with which we began. Our earlier machine semantics formed a monitored history by pairing an execution with a compatible monitor history. Does that construction survive the passage from the machine topos $\mathcal{E}_J$ to the temporal topos $\mathcal{B}$?

Yes, at the level of these pullbacks. Let $X \to E$ be a machine's event map and $M \to E$ its timed monitor. The finite-limit preservation of L in (17) gives

$$L(X \times_E M) \cong LX \times_{LE} LM. \tag{25}$$

Similarly, a good-monitor monomorphism remains a monomorphism after applying L. Therefore an *already proved* factorization of initialized reachable histories through the safe monitored subobject continues to be a factorization after the comparison.

It is tempting to read more into this than it says. The functor L preserves the finite-limit constructions just used, but left exactness alone does not tell us that it preserves arbitrary temporal implications, exponentials, or future quantification. That is why we calculated the oscillator's response witnesses directly in the previous section. For general variable-duration hybrid runners, the comparison with the internal temporal modalities is still an open problem.

15 What is proved, and what remains

We began with an ordinary programming-language picture: a source program requests an observation, and a runner implements that request using another set of operations. By following the execution rather than just its final answer, we found three increasingly informative ways of describing the same interaction: a tree of implementing operations, a sheaf of temporal histories, and a timed machine whose behaviour can be checked against a deadline.

For clarity, here is the boundary between the results established in this interlude and the questions we have only prepared for.

Claim	**Status**
One-read Mealy runners unfold to path sheaves; serial pullback law	Proved
Positive well-founded runner substitution and nested execution	Proved (decorated)
Event-clocked serialization of positive well-founded executions	Proved up to state normalization
Amortised duration certificates compose as $(rq, q\Phi + \Psi)$	Proved
Bounded response for completion events	Proved under hypotheses
Monitor pullbacks commute with wiring	Proved
Continuous sampled runner has a suspension trajectory sheaf	Constructed under hypotheses
Temporal sheafification of the alternating oscillator	Explicitly calculated
Sharp constant-duration witness versus oscillator monitor test	Proved extensionally
Clock-indexed Time response and intrinsic modal-monitor equivalence	Open
General Sweedler measuring existence in Grothendieck toposes	Separate open problem

There are two natural continuations. The first is operational: allow nondeterministic effects, implementations that may perform zero target reads, or computations that may block or diverge. Then we would have to distinguish possible from guaranteed response and account for silent completions and Zeno executions. A double category of *timed certified runners* looks attractive, but it would need a careful choice of 2-cells and a faithful account of execution phases.

The second is foundational. We still lack a satisfactory general theorem saying when the enriched accessible endofunctors of a Grothendieck topos admit the duoidal Sweedler adjunctions we want. Restricting to polynomial functors, using systems of arities, and applying presentability methods are possible routes, not yet results of this chapter. The temporal constructions give us explicit tests that such a theorem ought to explain.

There is also a meeting point between these two directions. The clock-slice equivalence $\mathrm{Sh}(\mathsf{IR}) \simeq \mathcal{B}/\mathsf{Time}$ suggests a dependent approach in which a runner and its state are indexed by a marked temporal world. That would bring the present work back to the dependent Sweedler thread, and perhaps explain why the states of temporal systems are often best understood through local behaviour near a chosen time.

Place alongside the two research papers. Paper I [6] explains the algebraic side of finite operational programs, graded comodels, and temporal Chu evaluation; Paper II [7] explains how residual implementations retain and compose cost and ownership certificates. The present Interlude contributes the physical-time semantics not established in those papers: trajectory sheaves, synchronization, continuous suspension, monitoring, and a bounded-response calculation in a temporal topos. A unifying realization theorem, functorial in graded certified runners and compatible with both Chu evaluation and temporal sheaf wiring, is still a target rather than an established result.

References

[1] P. Schultz and D. I. Spivak, *Temporal Type Theory: A Topos-Theoretic Approach to Systems and Behavior*, Progress in Computer Science and Applied Logic, vol. 29, Birkhäuser, 2019. doi:10.1007/978-3-030-00704-1; arXiv:1710.10258.

[2] P. Schultz, D. I. Spivak, and C. Vasilakopoulou, "Dynamical Systems and Sheaves," *Applied Categorical Structures* (2020), doi:10.1007/s10485-019-09565-x; arXiv:1609.08086.

[3] J. C. Willems, "Paradigms and puzzles in the theory of dynamical systems," *IEEE Transactions on Automatic Control* **36** (1991), no. 3, 259–294. doi:10.1109/9.73561.

[4] D. J. Myers, *Categorical Systems Theory*, draft manuscript, §6.2. Online manuscript.

[5] S. Libkind and D. J. Myers, "Towards a double operadic theory of systems," preprint (2025). arXiv:2505.18329.

[6] *Sweedler Runners for Temporal Systems, I: From Graded Operational Syntax to Temporal Behaviour. Polynomial Moore Trees, Bounded Comodels, and Chu Evaluation*, companion research draft (October 2026).

[7] *Sweedler Runners for Temporal Systems, II: Resource-Certified Implementations and Ownership. Cost, Temporal Evaluation, and Potentials*, companion research draft (October 2026).