CATEGORICAL FOUNDATIONS / RESEARCH

Semantic Towers and Authority-Preserving Multilevel Computations

Categorical Foundations for Inspectable Formal Artefacts and Delegating Agents

ABSTRACT

Large generated formal developments and delegating agent systems share a problem: a concrete computation must remain related to an intelligible specification while its representation, executor, or degree of detail changes. We develop an elementary categorical account based on partial interpretations defined on full subcategories of observable states. We distinguish semantic soundness, representation totality, source-lifting completeness, fullness, progress, and recoverability. Interpretations compose; we prove sufficient safe-lifting hypotheses for observability to compose, and an explicit counterexample shows why ordinary completeness is insufficient. Semantic towers are then diagrams of these interpretations, with equality of domains included in coherence. Proof inspection adds checked evidence and dependency manifests; agent systems add task contracts and authority requirements. A separate confinement theorem bounds concrete execution prefixes, and a migration theorem transports invariant evidence without claiming equivalence of future executions. Worked models cover generated proof inspection, delegated payments, agent migration, and a transactional industrial controller. Fibrations, profunctors, and double categories are evaluated as optional extensions rather than assumed foundations. The resulting guarantees are conditional on explicit semantic, scheduling, and enforcement boundaries.

Reading guide

Foundations use a violet rule. Worked examples have a warm stone background. Margin notes identify assumptions, distinctions, and limits; they add no unstated hypotheses.

1 Meaning, inspection, and control

A checked proof can be difficult to inspect, and a well-specified task can be difficult to delegate. These difficulties are related. In each case, a person or component relies on an abstract account while another computation manipulates a more elaborate representation. The question is not simply whether the representation produces an acceptable final answer. It is also whether its current meaning can be recovered, whether that meaning survives a change of implementation, and whether a change of representation changes what the executor is permitted to do.

The intellectual starting point is François-René Rideau's thesis programme, *Reconciling Semantics and Reflection: Formalisms, Protocols, Applications and Architecture*. Its core account of partial interpretations, observable states, semantic towers, and runtime control was presented publicly in 2018 [1]. We develop the needed definitions and arguments here, while making recovery composition and authority preservation explicit proof obligations. We neither identify recoverability with eventual progress nor infer security from semantic correctness.

An interpretation goes from a concrete computation to an abstract one. Realizing an abstract state goes in the opposite direction, but generally offers many choices or none. It is not an inverse functor. A controller, meanwhile, can select or replace a realization; this is a third kind of activity. Generating a proof or program, interpreting its execution, and authorizing changes to it should therefore remain distinct even when one software component performs all three.

Our common foundation is deliberately small: categories of finite computations, functors recording effects, and partial interpretations with explicit domains. Proof inspection specializes this foundation with obligations and evidence. Agents specialize it with task contracts, grants, and mediated actions. Neither specialization changes what an interpretation means. The useful theorems concern the transport of properties through specified boundaries; they do not establish that an informal requirement was formalized correctly or that an implementation's hardware model describes the physical world.

Standard categorical terminology follows Mac Lane [2]. Computation and indexing categories are small relative to a fixed universe; categories of these structures may live in a larger universe. We write gf for $g \circ f$, with f executed first. An inclusion is drawn with a hook, a total functor with a solid arrow, and a partial interpretation with a dashed arrow. Dashed arrows abbreviate explicit full-domain presentations; they are not missing ordinary morphisms. Equality of paths is strict unless an isomorphism or another comparison is expressly supplied.

2 Computation categories and action labels

A state description alone does not determine what counts as the same computation. Two executions can have equal endpoints while writing different audit records, consuming different inputs, or exercising different permissions. Such distinctions must be represented before a preservation theorem can mention them.

What do labels retain?
Effects omitted by a label map are outside the corresponding preservation claim.

> **Definition 2.1** (Labelled computation)**.** A labelled computation is a category C, an *action category* E_C, and a functor $\lambda_C : C \to E_C$. Objects of C are states; arrows are finite admissible computations. The image $\lambda_C(f)$ records the effects retained by this particular semantics. An arrow is *silent* when its label is an identity.

Identities describe empty computations, and composition describes sequential execution. Silence is relative to a label functor: a silent arrow can change memory, consume time, or require authority that this functor omits. Action categories can have multiple objects, recording interaction modes. A one-object action category is a monoid. For an alphabet Σ, its free word monoid Σ^* gives a familiar example, with concatenation ordered by execution.

A labelled transition system starts with a directed multigraph G. Its free category $\mathrm{Path}(G)$ has finite paths as arrows, including empty paths. Assigning labels to generating edges extends uniquely to a functor into an action category, provided the objects and edge endpoints match. If executions are identified, the identification must be a congruence on parallel paths: it must respect pre- and post-composition. Labels descend to the quotient only if equivalent paths have equal labels. Merely declaring paths equivalent because they have the same final state may erase precisely the effects under investigation.

The category obtained by identifying all parallel arrows is thin and hence describes a preorder. It is a partial order only after antisymmetry has been established or mutually reachable objects have been identified. We use thin categories where existence of a transition suffices, and path categories where the trace matters.

Infinite runs are not arrows of this finite-computation category. To discuss scheduling and liveness we separately specify a set of generating steps and a class of admissible, or fair, runs. This additional information is not recoverable from an arbitrary quotient category. Similarly, probabilities and durations require additional structure; they do not follow from the category axioms.

Example 2.2 (A transaction between checkpoints)**.** An abstract transfer is one arrow from a balanced ledger to another balanced ledger. A concrete implementation may debit, record an outstanding obligation, and then credit. The intermediate memory state need not denote an abstract balanced ledger. Nevertheless, the complete concrete path has observable endpoints and must account for the transfer and any external effects. Partial interpretation is appropriate; deleting the debit–credit path from the semantics is not.

3 Partial interpretations and implementation properties

3.1 Observable domains and soundness

A full domain

Every path between observable endpoints is included, even if its intermediate states are hidden.

Choosing an abstraction involves choosing when its descriptions apply. Following Rideau [1, Section 2], we make this choice at objects and retain every computation between the chosen objects.

Definition 3.1 (Partial interpretation)**.** A partial interpretation $\Phi : C \dashrightarrow A$ is a pair (O_Φ, ϕ), where O_Φ is the full subcategory of C on a specified subset of $\mathrm{Ob}\, C$, and $\phi : O_\Phi \to A$ is a functor. Its objects are called *observable*. Write $j_\Phi : O_\Phi \hookrightarrow C$ for the inclusion.

The presentation and its abbreviation are displayed in (1). Only the solid legs are functors defined on their entire source; the dashed leg denotes the pair.

$$\begin{array}{ccc} O_\Phi & \xrightarrow{\ \phi\ } & A \\ {\scriptstyle j_\Phi}\downarrow & \nearrow_{\Phi} & \\ C & & \end{array} \tag{1}$$

Fullness of O_Φ says that an arrow is in the domain exactly when both endpoints are observable. Its internal factorization can pass through unobservable states. This forces the abstraction to account for every admissible concrete behavior between checkpoints. If some executions are to be ruled out by policy, that restriction belongs in the concrete computation category, not in an ad hoc removal of arrows from an otherwise full domain.

Functoriality is the basic semantic soundness requirement. It preserves source, target, identity, and composition; in particular the following square of functions between hom-sets commutes for observable x, y, z:

$$\begin{array}{ccc} O_\Phi(y,z) \times O_\Phi(x,y) & \xrightarrow{\ \circ\ } & O_\Phi(x,z) \\ {\scriptstyle \phi\times\phi}\downarrow & & \downarrow{\scriptstyle \phi} \\ A(\phi y, \phi z) \times A(\phi x, \phi y) & \xrightarrow[\ \circ\ ]{} & A(\phi x, \phi z) \end{array} \tag{2}$$

This condition says more than merely matching reachable endpoints. It records the agreement between interpreting a composite and composing its interpretations.

For labelled computations we additionally specify a functor $h : E_C \to E_A$ and require

$$\begin{array}{ccc} O_\Phi & \xrightarrow{\ \phi\ } & A \\ {\scriptstyle \lambda_C\, j_\Phi}\downarrow & & \downarrow{\scriptstyle \lambda_A} \\ E_C & \xrightarrow[\ h\ ]{} & E_A \end{array} \tag{3}$$

Thus $\lambda_A \phi = h \lambda_C\, j_\Phi$. The chosen action abstraction h may hide effects, but the theorem then concerns the retained effects. A functional-correctness label map that hides a privileged action cannot simultaneously be used as evidence that the action was authorized.

3.2 Four distinct realization properties

Soundness constrains concrete executions. Realization properties ask what abstract requests can be implemented. The order of their quantifiers matters.

Definition 3.2 (Totality, completeness, and fullness)**.** An interpretation has *total interpretation domain* if $O_\Phi = C$. It is *representation-total* if for every $a \in A$ there is $o \in O_\Phi$ with $\phi(o) = a$. It is *complete* if, for each $o \in O_\Phi$ and $u : \phi(o) \to a$, there exist $o' \in O_\Phi$ and $f : o \to o'$ such that $\phi(o') = a$ and $\phi(f) = u$. It is *full* if every map $O_\Phi(o, o') \to A(\phi o, \phi o')$ induced by ϕ is surjective.

Quantifier order

Completeness fixes the source. Fullness fixes both endpoints. Totality asks whether representations exist.

We use strict equality in these definitions. Essential surjectivity and lifting up to specified isomorphism are legitimate alternatives, but require transport data. Representation totality is the property called totality in the motivating thesis; it does not assert that every concrete state is interpretable.

Completeness fixes the current concrete source and leaves the concrete target to the implementation. Equivalently, for each o and a, the following function is surjective:

$$\coprod_{\substack{o' \in O_\Phi \\ \phi(o') = a}} O_\Phi(o, o') \xrightarrow{f \mapsto \phi(f)} A(\phi(o), a) \tag{4}$$

There is no universal property or uniqueness of a lift. In particular completeness is not the definition of an opfibration. Fullness instead fixes both representations, which can require undoing an irreversible implementation choice.

Proposition 3.3. *Fullness and representation totality together imply completeness. Completeness and representation totality together need not imply fullness.*

Proof. Given $u : \phi(o) \to a$, use representation totality to choose o' over a, then fullness to lift u with these endpoints. For the converse, map the discrete category on two objects to the terminal category. Every required lift from a fixed source is its identity, so the functor is complete and representation-total. It is not full between the two distinct objects. □

A full complete inclusion of one object into a discrete category on two objects is not representation-total. A total-domain functor need not be representation-total either. Empty domains make soundness and completeness vacuous. A useful implementation contract must therefore include an initial realization or an appropriate coverage requirement.

3.3 Composition

Suppose $\Phi : C \dashrightarrow A$ and $\Psi : A \dashrightarrow B$. A state can be interpreted twice precisely when it lies in the first domain and its first interpretation lies in the second. Define

$$D = C\big[\{c \in \mathrm{Ob}\, O_\Phi \mid \phi(c) \in \mathrm{Ob}\, O_\Psi\}\big], \qquad \Psi\Phi = (D, \psi\phi|_D).$$

Here $C[S]$ denotes the full subcategory on S. The square in (5) is a strict pullback in **Cat**; fullness of O_Ψ ensures that ϕ restricts to the indicated functor.

$$\begin{array}{ccccc} D & \xrightarrow{\phi|_D} & O_\Psi & \xrightarrow{\psi} & B \\ \downarrow & \lrcorner & \downarrow{\scriptstyle j_\Psi} & & \\ O_\Phi & \xrightarrow[\phi]{} & A & & \\ {\scriptstyle j_\Phi}\downarrow & & & & \\ C & & & & \end{array} \tag{5}$$

Proposition 3.4. *Small categories and these partial interpretations form a category* $\mathbf{Par}_{\mathrm{full}}$. *Composition preserves soundness, representation totality, completeness, and fullness, each whenever both factors have the property. Label-compatible interpretations compose with action map* $h_\Psi h_\Phi$.

Proof. For three factors, either association selects exactly the objects c such that $c \in O_\Phi$, $\phi(c) \in O_\Psi$, and $\psi\phi(c) \in O_\Xi$. The full subcategories are equal, and the functors agree on objects and arrows by ordinary associativity. The identity is (C, id_C). Soundness follows by composing restricted functors.

For representation totality, realize a target first in the middle category and then in the source. For completeness, lift the requested arrow through ψ from the prescribed middle state, then lift that arrow through ϕ from the prescribed source. The resulting endpoint lies in D. For fullness the source and target in D are already fixed; apply fullness first to ψ and then to ϕ. Finally, pasting the label squares yields the asserted label equation. □

This is a concrete instance of the partial-map approach developed systematically by Cockett and Lack [3]: domains are represented by inclusions and composition restricts by pullback. We prove the specialized construction directly. No choice of 2-cells is needed here; later weak comparisons require additional structure.

4 Recovery, progress, and the composition of observability

An execution can be interrupted between checkpoints. Recovering a meaning from that interruption is different from observing the execution that the scheduler would have chosen to continue.

Recovery is existential

A safe return can exist while a scheduler postpones it forever. A strategy or time bound is extra data.

Definition 4.1 (Safe recovery)**.** Choose a wide subcategory $S_C \subseteq C$ of *safe arrows*. Let

$$\mathrm{Reach}_C(O) = \{x \in \mathrm{Ob}\, C \mid \exists o \in O,\ \exists f : o \to x\}.$$

The interpretation Φ is *observable relative to* S_C when, for every $x \in \mathrm{Reach}_C(O_\Phi)$, there exist $o' \in O_\Phi$ and $\sigma : x \to o'$ in S_C.

The word safe specifies which recovery operations a contract permits. It is not a theorem that these operations are physically safe, silent, cost-free, or non-advancing. Identities and composition belong to S_C, so successive permitted recovery operations remain permitted. In (6), f is given and o', σ are required witnesses. The triangle is in C. Its outside arrow lies in O_Φ and has interpretation $\phi(\sigma f)$; neither $\phi(x)$ nor $\phi(f)$ need exist.

$$o \xrightarrow{f} x \xrightarrow{\exists\, \sigma \in S_C} o' \qquad (o, o' \in O_\Phi) \tag{6}$$
$$\sigma f$$

This definition is existential. An environment may prevent the recovery routine from executing, and an unfair scheduler may postpone it forever. A strategy that forces recovery against an adversary is a stronger game-theoretic property. A time bound and a constructive recovery algorithm are also additional obligations.

4.1 Liveness and co-liveness

Classical liveness concerns properties of whole executions [4]. The following checkpoint condition is deliberately relative to an explicit run model. For precision, choose progress gradings $p_C : C \to B\mathbb{N}$ and $p_A : A \to B\mathbb{N}$, where $B\mathbb{N}$ is the one-object category of natural numbers under addition. A computation advances when its grading is positive. This is a restrictive but useful explicit progress model. Its positive arrows form a two-sided ideal,

while zero-progress arrows form a wide subcategory. There is no required equality between zero-progress arrows and safe arrows.

For a specified generating-step and fair-run model, call Φ *checkpoint-live* when every fair infinite run beginning at $o_0 \in O_\Phi$ and having unbounded concrete progress has observable indices $n_0 = 0 < n_1 < \cdots$ for which the abstract progress of the interpreted prefixes is unbounded. The prefixes between such indices lie in the full observable domain. This is a particular strong liveness condition, not a definition of every possible liveness notion. It makes no assertion about infinite runs of bounded concrete progress; excluding such divergence needs an additional condition.

A state is *progress-terminal* when no positive-progress arrow leaves it. Call Φ *co-live* when every reachable concrete progress-terminal state is observable and has progress-terminal abstract image. This property concerns stopping, whereas checkpoint liveness concerns continuing. Neither is implied by functoriality. Completeness alone does not yield co-liveness: the lift of a positive abstract arrow could have zero concrete progress unless a progress-compatibility condition excludes this.

An observable system can still loop forever between unobservable states while a safe exit remains available. Conversely, with the safe class restricted to identities, an execution may repeatedly visit checkpoints and satisfy liveness without admitting safe recovery at an intermediate state. Progress and recoverability answer different questions. We make no general composition theorem for fair-run liveness without specifying how runs and fairness project through each level.

4.2 The missing lifting hypothesis

Counterexample 4.2. Let $C = A$ be the thin category $0 \to 1 \to 2$, and let Φ be its identity with full domain. Take S_C to contain only identities. Let $O_\Psi = A[\{0, 2\}]$ and map it isomorphically to the two-object chain B. Take S_A to contain identities and $1 \to 2$. Both interpretations are observable. The lower one is even complete, full, and representation-total. Their composite has observable objects $0, 2$, but the reachable object 1 has no S_C-arrow to either. Thus its observability fails.

The upper recovery step exists, but there is no *safe* lower realization of it. This identifies the extra hypothesis rather than merely demonstrating a failure.

The lifting obligation
The upper safe step must have a safe realization from the actual recovered lower state.

Definition 4.3 (Safe completeness and preservation)**.** Given S_C, S_A, an interpretation is *safe-complete* if every $s : \phi(o) \to a$ in S_A, with $o \in O_\Phi$, lifts to $u : o \to o'$ in S_C with $o' \in O_\Phi$ and $\phi(u) = s$. It is *safe-preserving* if $\phi(S_C \cap O_\Phi) \subseteq S_A$.

Theorem 4.4 (Compositional observability)**.** *Let $\Phi : C \dashrightarrow A$ and $\Psi : A \dashrightarrow B$ be observable relative to S_C and S_A, respectively. If Φ is safe-complete, then $\Psi\Phi$ is observable relative to S_C.*

Proof. Let $d \in D$ and $f : d \to x$. By observability of Φ, choose $\sigma : x \to y$ in S_C with $y \in O_\Phi$. The arrow σf has endpoints in O_Φ. Consequently $\phi(y)$ is reachable from $\phi(d) \in O_\Psi$. Observability of Ψ gives $s : \phi(y) \to a'$ in S_A with $a' \in O_\Psi$. Safe completeness of Φ, applied at this particular y, gives $u : y \to y'$ in S_C with $\phi(u) = s$ and $\phi(y') = a'$. Hence $y' \in D$, and $u\sigma : x \to y'$ is safe. The lifting square of this argument is shown in (7). □

$$\begin{array}{ccc} L_y & \hookrightarrow & O_\Phi(y,-) \\ {\scriptstyle\phi}\downarrow & & \downarrow{\scriptstyle\phi} \\ S_A(\phi y,-) & \hookrightarrow & A(\phi y,-) \end{array} \tag{7}$$

$$x \xrightarrow{\sigma \in S_C} y \xrightarrow{u \in S_C} y' \qquad \phi(u) = s, \quad y' \in D.$$

In this diagram the square is in **Set**. Write $O_\Phi(y,-)$ and $A(\phi y,-)$ for the disjoint unions of hom-sets with the indicated source, and let L_y consist of safe arrows in $O_\Phi(y,-)$ whose images are safe. The bottom left set contains all safe arrows from ϕy. Thus the left function is well defined without assuming safe preservation, and its surjectivity states precisely the lifting obligation. The concrete composable arrows beneath it are in C.

Corollary 4.5. *Safe-preserving, safe-complete, observable interpretations compose, with identities, among categories carrying chosen safe subcategories.*

Proof. Safe preservation follows by composition. To lift a safe arrow in the final category, safe completeness of the upper map supplies a safe middle arrow, and that of the lower map supplies a safe source arrow. Its endpoint lies in the composite domain. Observability follows from Theorem 4.4. Identities have all three properties. □

For bounded recovery, suppose nonnegative costs satisfy $\ell(gf) \le \ell(f) + \ell(g)$. If lower recovery costs at most b_Φ, upper recovery at most b_Ψ, and each safe upper arrow of cost at most t has a chosen lower lift of cost at most $L(t)$, then the constructed recovery costs at most $b_\Phi + L(b_\Psi)$. This follows directly from the two witnesses in the proof. The bound includes the cost of lifting; simply adding two unrelated layer bounds is unjustified.

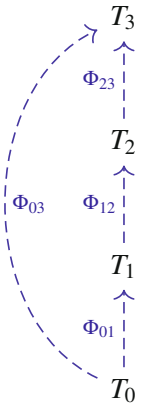
5 Semantic towers and alternative paths

A hierarchy is useful only if successive interpretations agree about what is being interpreted. The choice of observable domains is part of this agreement, not a detail that can be suppressed after drawing a tower.

Coverage is semantics
Two paths that agree on an overlap can still disagree about which states they interpret.

> **Definition 5.1** (Strict semantic tower)**.** Let L be a small category of views. A strict semantic tower is a functor $T : L \to \mathbf{Par}_{\mathrm{full}}$. Thus a view i has a computation category T_i, and an arrow $u : i \to j$ has a partial interpretation $\Phi_u : T_i \dashrightarrow T_j$. We require $\Phi_{1_i} = 1_{T_i}$ and $\Phi_{vu} = \Phi_v \Phi_u$, including equality of domains.

For a linearly ordered set of levels, the familiar picture is (8). Arrows point toward the more abstract descriptions. The direct dashed arrow has exactly the domain obtained by the successive restrictions; it is not declared total on the bottom level.



$$\Phi_{03} = \Phi_{23}\Phi_{12}\Phi_{01}. \tag{8}$$

There is no distinguished universal lowest level. A level is concrete or abstract only relative to an interpretation. Likewise, the top formal category is a specification accepted for a purpose, not human intention made infallible by notation.

A free category on a graph of views permits several paths with the same endpoints. Being a functor from that free category does not force these paths to agree. To demand agreement, specify relations between paths in the index category and verify that their interpretations satisfy the relations. In particular, for a commuting square in L, its image must satisfy

$$\begin{array}{ccc} T_1 & \xrightarrow{\Phi_v} & T_3 \\ {\scriptstyle \Phi_u}\uparrow & & \uparrow{\scriptstyle \Phi_s} \\ T_0 & \xrightarrow[\Phi_r]{} & T_2 \end{array} \tag{9}$$

where $\Phi_v\Phi_u = \Phi_s\Phi_r$ is an equality of partial interpretations. It means equality of the two inverse-image domains and equality of the resulting functors there. Agreement on the intersection alone is weaker: two maps with disjoint domains agree there vacuously and still offer incompatible coverage.

Example 5.2 (A coherent diamond)**.** For categories X, Y, Z, take the bottom view $X \times Y \times Z$, intermediate views $X \times Y$ and $X \times Z$, and top view X. Forgetting the unused factors gives two strictly equal total projections to X. If instead the first intermediate view is observable only when the Y-component lies in a chosen subset, while the second requires a separate condition on Z, the composite domains generally differ. The two paths still return the same X-component wherever both are defined, but no strict commuting diamond exists until their coverage is reconciled.

Sometimes a natural isomorphism is more appropriate than strict equality. On an explicitly chosen common full domain D, one can require

$$\eta : (\phi_v\phi_u)|_D \;\Rightarrow\; (\phi_s\phi_r)|_D$$

to be a natural isomorphism. Components and their naturality squares then express the comparison. For three alternative paths one must also require the direct comparison to equal the composite comparison; larger diagrams need compatible coherence equations. Even these equations are not enough to justify unrestricted composition of partial comparisons: a later interpretation may not be defined on an isomorphic replacement object. Replete observable domains, closed under isomorphism, are one possible additional hypothesis. We use strict towers for the principal results and leave a general weak partial theory open.

A refinement contract records the domain, interpretation, label map, safe class, and whichever realization, recovery, progress, evidence, or authority properties are required. Each field is either justified or identified as an assumption. Theorem 4.4 is a rule for composing particular fields, not a rule that every desirable property of a layer composes automatically.

6 Inspectable formal artefacts and evidence

Inspection requires a stable claim and a way to justify it. The separation of untrusted production from checked evidence follows the proof-carrying-code discipline [5]; we additionally track intermediate checkpoints. An abstract label such as “verified” is insufficient unless the proposition, assumptions, and checked evidence are recoverable. We distinguish the computation being inspected from the mathematical evidence attached to its states.

Evidence has direction
Runtime invariants move forward. Solutions of refined proof obligations move backward.

The construction below is the covariant category of elements; its lifting direction is that of an opfibration [6, Chapter 9].

Definition 6.1 (Evidence-bearing computation)**.** Let W be a workflow category and $E : W \to \mathbf{Set}$ a functor. Its category of elements $\int E$ has objects (w, e) with $e \in E(w)$, and arrows $(w, e) \to (w', e')$ given by $f : w \to w'$ satisfying $E(f)(e) = e'$.

Proposition 6.2. *This is a category. Its projection to W is a discrete opfibration: every $f : w \to w'$ has exactly one lift with prescribed source (w, e).*

Proof. The identity equation follows from $E(1_w) = 1_{E(w)}$. If $E(f)(e) = e'$ and $E(g)(e') = e''$, then $E(gf)(e) = e''$, so composition is inherited from W. The unique lift of f from (w, e) has target $(w', E(f)(e))$; an arrow above f is determined by that equation. □

The functorial evidence transformer is an assumption of this construction. Arbitrary operational transitions do not acquire proofs merely by being put in W. If a transition requires a newly checked certificate, that certificate must be part of the arrow data, with the evidence transformer defined from it. A missing or rejected certificate yields no trusted arrow. Identity and composition laws for these transformers must then be proved.

Given evidence functors $E_C : C \to \mathbf{Set}$ and $E_A : A \to \mathbf{Set}$, an evidence-preserving interpretation can carry a natural transformation $\tau : E_C j_\Phi \Rightarrow E_A \phi$. For each observable $f : o \to o'$, the square

$$\begin{array}{ccc} E_C(o) & \xrightarrow{E_C(f)} & E_C(o') \\ {\scriptstyle \tau_o}\downarrow & & \downarrow{\scriptstyle \tau_{o'}} \\ E_A(\phi o) & \xrightarrow[E_A(\phi f)]{} & E_A(\phi o') \end{array} \tag{10}$$

requires that transport before interpretation agrees with interpretation before transport. All four objects are sets of evidence; this is an ordinary naturality square.

A checkpoint can now record a current state x, a safe recovery $\sigma : x \to o$, its abstract state $a = \phi(o)$, checked evidence for the stated claim at a, and a manifest Δ of assumptions, unresolved obligations, versions, and external attestations. A claim with unresolved premises is a conditional claim. It is not silently promoted to an unconditional theorem. Observability supplies the recovery component, but supplies neither the evidence nor the manifest.

Inspection is also not necessarily reversible. Recovery might complete a transaction rather than undo it. If a user expects to resume as though inspection had not intervened, the contract must compare the permitted continuations, including any effects of recovery. No inverse of σ is assumed.

6.1 Worked example: inspecting a generated induction proof

The backward solution transformer is the validation component of the LCF tactic discipline [7]. The example abstracts that discipline; it does not claim an implemented kernel adapter.

WORKED EXAMPLE / PROOF INSPECTION

Fix an environment Γ and a theorem

$$P \equiv \forall n \in \mathbb{N}, \quad 2\sum_{i=0}^{n} i = n(n+1).$$

Let an abstract proof state be a finite list Ω of obligations. An arrow $\Omega \to \Omega'$ is a certified reduction: a function sending proofs of all obligations in Ω' to proofs of all obligations in Ω. Composition is function composition in that backward direction. Equivalently, choose actual proof sets $\mathrm{Sol}(\Omega)$ and take these solution-transforming functions as arrows. Identities and associativity are immediate, and $\mathrm{Sol} : A^{\mathrm{op}} \to \mathbf{Set}$ is contravariant. This variance describes proof search: later obligations supply evidence for earlier ones.

WORKED EXAMPLE / PROOF INSPECTION (continued)

The generated induction plan reduces $[P]$ to a base obligation and an induction-step obligation. The base simplifies to $0 = 0$. In the step, the induction hypothesis supplies $2\sum_{i=0}^{n} i = n(n+1)$; adding $2(n+1)$ yields

$$2\sum_{i=0}^{n+1} i = n(n+1) + 2(n+1) = (n+1)(n+2).$$

The reduction to the empty obligation list therefore gives a function from its singleton solution set to $\mathrm{Sol}([P])$, hence a proof of P.

A concrete category can consist of certified proof-plan states: elaborated syntax, partial proof trees, substitutions, and metavariable constraints. Its arrows are certified plan refinements with the corresponding solution transformers. Raw model proposals are candidates for these arrows; unsuccessful checking is outside the accepted category. Choose as observable those states with a resolved, exportable obligation interface, and interpret each by its obligation list and induced solution transformer. To obtain a functor, any quotient identifying concrete derivations must identify only equal induced transformers. Fullness of the observable domain ensures that a long elaboration with hidden intermediate constraints still has a certified composite meaning.

A reviewer can inspect the induction plan, then expand the arithmetic step, then inspect the kernel certificate. The reduction direction explains why these views preserve the theorem without requiring every temporary metavariable to have a domain-level explanation. It does not prove that the chosen theorem expresses the user's original intent. Nor do we assume that an arbitrary proof assistant can always finish elaboration safely: an inspection implementation needs a recovery operation, such as restoring the last certified checkpoint, and a proof that this operation respects its declared semantics.

A finite dependency graph can make Δ and the transitive dependency cone computable. It need not make that cone small. Changes to dependencies justify conservative rechecking; they do not prove that every affected statement has become false or that every syntactic dependency is mathematically necessary.

7 Tasks, delegation, and interaction

Acceptance and success

A valid handoff establishes a starting contract. It does not supply a successful strategy.

An agent's language model, planner, or human operator can be opaque while the boundary at which its actions are accepted is precise. We model a task by a computation category A, an initial-state predicate P, a success predicate Q, and a designated set of acceptable failure or escalation states. A successful finite execution is an arrow $a \to a'$ with $P(a)$ and $Q(a')$. This is a partial-correctness specification in the sense of Hoare [8]; a claim that some executor will produce such an arrow requires a strategy or termination argument.

A handoff specifies the task and current abstract state, evidence for its precondition, a receiver realization, an attenuated grant, and validity conditions such as policy version, expiry, and environment assumptions. Receipt of a task packet is not evidence of its precondition. A proof or explicitly trusted attestation supplies that evidence. Proof-carrying authentication provides the precedent for requests accompanied by checkable authorization derivations [9]; task preconditions remain separate claims. Likewise, acceptance means that the receiver has the required initial representation and enforcement conditions; it need not mean that success is guaranteed.

For sequential decomposition, choose intermediate predicates $R_0 = P, R_1, \ldots, R_n = Q$. Each accepted component must send an R_{i-1}-state to an R_i-state, or to a declared failure state. Composition then transports the intermediate evidence and gives the usual partial-correctness argument. A return from a delegate consists of an observable result and evidence for the

requested postcondition. Natural-language declarations of success do not create morphisms in the trusted evidence category.

A handoff between heterogeneous representations uses the observation–interpretation–reimplementation pattern formalized in Section 10. This also applies when a human takes over a task, provided the human interface has an explicit task-state model. It is not a mathematical claim about the reliability of human judgment or of a particular model.

7.1 Parallel work and the independence assumption

For genuinely independent computations C, D, the product category represents paired states and paired computations. Its elementary interaction square is

$$\begin{array}{ccc} (c,d) & \xrightarrow{(f,1_d)} & (c',d) \\ {\scriptstyle (1_c,g)}\downarrow & & \downarrow{\scriptstyle (1_{c'},g)} \\ (c,d') & \xrightarrow[(f,1_{d'})]{} & (c',d') \end{array} \tag{11}$$

Both paths equal (f, g) by the componentwise definition of composition. Thus the square proves commutation in this model. It does not prove that two real agents touching a shared account commute. Shared state, message delivery, races, locks, and ordering constraints require a joint interaction category in which this equation may fail. A synchronized model may use a pullback over a shared interface, but equality of interface arrows is itself a chosen synchronization discipline. Separation logic makes resource separation an explicit premise of local reasoning [10]; drawing a product does not discharge that premise.

If a scheduler chooses an interleaving, that scheduler is additional run data. Product structure alone neither supplies fairness nor guarantees that agents simultaneously reach observable checkpoints. In particular, individual recovery routines may interfere through shared resources even when each has an existence proof in isolation.

8 Authority-preserving refinement

Semantic detail and executable authority are independent. A concrete representation may reveal how to construct a command without conferring permission to issue it. Our delegation discipline follows the object-capability treatment of authority and robust composition [11]. We first treat reusable permissions, then account for consumable quantities in the payment example.

Two kinds of resource

Reusable permissions combine by joins. Consumable budgets need state or a separate resource semantics.

Definition 8.1 (Permission-labelled computation)**.** Let $(K, \leq, \vee, \bot)$ be a join-semilattice with least element. A requirement map on C is a map $\rho_C : \mathrm{Mor}(C) \to K$ satisfying

$$\rho_C(1_c) = \bot, \qquad \rho_C(gf) = \rho_C(f) \vee \rho_C(g).$$

An arrow is permitted by a grant k when $\rho_C(f) \leq k$.

Equivalently, ρ_C is a functor to the one-object category with monoid $(K, \vee, \bot)$. For example, $K = \mathcal{P}(\mathcal{U})$ consists of sets of primitive permissions. A trace requires the union of the permissions exercised by its steps. Any quotient of traces must respect this requirement map. Exact accumulation prevents a later step from cancelling the permission requirement of an earlier step.

Proposition 8.2. *For each k, the arrows permitted by k form a wide subcategory C_k. If $k' \leq k$, there is an inclusion $C_{k'} \hookrightarrow C_k$.*

Proof. Identities require $\bot$. If $\rho(f), \rho(g) \leq k$, their join is at most k. The second assertion follows by transitivity of the order. □

Attenuation is the choice of $k' \le k$. This is not an operation that augments the receiver's authority with its ambient credentials. Enforcement must ensure that the task executes under its effective delegated grant. Distinct signers, revocation, and consumable currency are not captured merely by taking a union of reusable permissions.

Definition 8.3 (Authority-preserving interpretation)**.** Let C, A carry requirements in K_C, K_A, and let $\alpha : K_A \to K_C$ preserve finite joins and bottom. An interpretation $\Phi : C \dashrightarrow A$ is authority-preserving relative to α when, for every $f \in O_\Phi$,

$$\rho_C(f) \le \alpha\big(\rho_A(\phi(f))\big). \tag{AP}$$

The inequality bounds the actual concrete requirement by the explicitly translated abstract requirement. It is a separate obligation from soundness. In the square (12), the arrows are functions of sets, and the square satisfies the pointwise inequality (AP), not an unlabelled assertion of strict commutativity.

$$\begin{array}{ccc} \mathrm{Mor}(O_\Phi) & \xrightarrow{\ \phi\ } & \mathrm{Mor}(A) \\ {\scriptstyle \rho_C}\downarrow & & \downarrow{\scriptstyle \rho_A} \\ K_C & \xleftarrow[\ \alpha\ ]{} & K_A \end{array} \qquad \rho_C \le \alpha\rho_A\phi. \tag{12}$$

This condition is intentionally strong. A concrete computation interpreted as an abstract identity must require no permission in this model. If recovery or administrative work needs a privilege, it must be represented in the abstract contract or covered by an explicitly granted additional reserve. It cannot disappear merely because an effect-label functor hides it.

Proposition 8.4. *Authority-preserving interpretations compose, with permission translation given by composition of the translations.*

Proof. For an arrow f in the composite domain, apply the lower inequality and then the upper one. Monotonicity of the lower translation gives $\rho_C(f) \le \alpha_\Phi\rho_A(\phi f) \le \alpha_\Phi\alpha_\Psi\rho_B(\psi\phi f)$. Composite translations preserve joins and bottom. □

The preceding inequality applies to observable arrows. The next result explains when it also controls the unobservable prefixes between checkpoints.

Why prefixes count

Extend a prefix to a checkpoint. Exact requirement accumulation prevents a hidden privilege from disappearing.

Theorem 8.5 (Confinement by observable completion)**.** *Let A_k be the category of abstract arrows permitted by k. Suppose $\Phi : C \dashrightarrow A_k$ is observable and authority-preserving relative to α, with exact requirement accumulation in C. Then every $f : o \to x$ beginning at $o \in O_\Phi$ satisfies $\rho_C(f) \le \alpha(k)$.*

Proof. By observability, extend f by a safe $\sigma : x \to o'$ with $o' \in O_\Phi$. Fullness of the observable domain gives $\sigma f \in O_\Phi$. Exact accumulation and (AP) yield

$$\rho_C(f) \le \rho_C(\sigma f) \le \alpha\big(\rho_{A_k}(\phi(\sigma f))\big) \le \alpha(k).$$ □

The theorem is about all modeled executions from observable starts, not only those chosen by a benevolent planner. Its application requires an initial checkpoint and a correct accounting of primitive requirements. To conclude deployment confinement, every executable path through tools, controllers, recovery routines, and credentials must be mediated by this model. This is the complete-mediation obligation of Saltzer and Schroeder [12]. A compromised enforcement boundary invalidates that hypothesis. Merely descending a tower is a mathematical interpretation or realization choice, not an executable transition that mints a grant.

Counterexample 8.6 (Soundness without confinement)**.** A free concrete loop requiring a payment permission can map functorially to the identity of a terminal abstract category. The interpretation is total and sound. It violates (AP), since the abstract identity requires $\bot$. Semantic correctness alone therefore cannot establish non-amplification of authority.

For independent agents with grants k_1, k_2, the product trace requires their join; a parent bound $k_1 \vee k_2 \leq k$ controls their reusable permissions. This conclusion does not say that two agents may each spend the same consumable budget.

9 Opacity, full abstraction, and information flow

A phrase such as "non-leaky abstraction" can hide several different obligations. Fix sets of observable representations and abstract states, together with explicit observation classes. Write $o \approx_C o'$ when every permitted concrete observer returns the same result on them, and similarly for $\approx_A$. A boundary is *fully abstract relative to these classes* when

$$o \approx_C o' \quad \Longleftrightarrow \quad \phi(o) \approx_A \phi(o').$$

This is the preservation-and-reflection criterion of full abstraction, originating in programming-language semantics [13], specialized to the observation classes just chosen. The observations may include contextual tests or execution outcomes, but their choice must be stated; category structure alone does not determine them.

Separate guarantees

Observational equivalence, control of actions, and secrecy of inputs are different obligations.

Representation opacity is a one-way condition: abstractly equivalent representations cannot be distinguished by a specified concrete client. One sufficient formulation is factorization $q = \bar{q}\phi$ on observable objects, with $\bar{q}$ invariant under $\approx_A$. Constancy on exact fibers yields a function on the image of ϕ, but does not determine observations on unrepresented objects or ensure invariance under $\approx_A$. It therefore does not establish contextual full abstraction.

Information-flow noninterference, introduced by Goguen and Meseguer [14], has a different shape. After choosing low-equivalence of initial states and a low trace observation, require low-equivalent initial states to yield equal low-observation sets of permitted runs. This is a set-of-traces variant; it is not the original deterministic formulation. Other variants compare distributions or scheduler-indexed runs. Hidden divergence, timing, termination, and scheduler knowledge must be addressed if included in the observer's power. This is a relation between executions, not merely an equation between current abstract states.

Capability confinement restricts which actions can occur. It does not prevent a read-authorized process from printing a secret if printing is also permitted. Conversely, two equivalent representations can both exercise an unauthorized privilege. An abstraction can therefore be fully abstract for one observation class while failing a relevant authority policy or a stronger information-flow policy. Migration transparency below is explicitly relative to the observations it preserves.

10 Migration, controllers, and instrumentation

10.1 Matching representations and transporting evidence

Suppose $\Phi : C \dashrightarrow A$ and $\Psi : K \dashrightarrow A$ implement the same abstract task. A migration begins by recovering $o \in O_\Phi$ from the current concrete state. It then chooses $k \in O_\Psi$ representing the recovered meaning. Representation totality of Ψ suffices to find such a k for every abstract object, but existence over the particular recovered object is all a single migration needs. Completeness is stronger and concerns subsequent abstract arrows.

State is not a future

Matching representations transports invariant evidence. Matching continuations requires simulation.

Definition 10.1 (Category of matching representations)**.** Let $M(\phi, \psi)$ have objects (o, k, θ), where $o \in O_\Phi$, $k \in O_\Psi$, and $\theta : \phi(o) \xrightarrow{\sim} \psi(k)$ is an isomorphism in A. An arrow to (o', k', θ') is a pair $(f : o \to o', g : k \to k')$ satisfying the commuting square

$$\begin{array}{ccc} \phi(o) & \xrightarrow{\phi(f)} & \phi(o') \\ {\scriptstyle\theta}\downarrow{\scriptstyle\sim} & & {\scriptstyle\sim}\downarrow{\scriptstyle\theta'} \\ \psi(k) & \xrightarrow[\psi(g)]{} & \psi(k') \end{array} \tag{13}$$

This is the iso-comma construction, obtained from the ordinary comma category [2, Chapter II] by restricting its comparison arrows to isomorphisms. Identities satisfy the square, and pasting two squares gives the equation for the componentwise composite, so it is a category. The projections to the two observable categories are functors. Reimplementation chooses an object of this category; it is not an application of a symbol Ψ^{-1} pretending to be a function.

An accepted handoff can therefore be recorded as

$$H = (x, \sigma, o, k, \theta, e, \kappa, \Delta; Q),$$

where $\sigma : x \to o$ is safe, $(o, k, \theta) \in M(\phi, \psi)$, e is checked precondition evidence at $\phi(o)$, κ is the effective target grant, Δ records the validity assumptions, and Q is the requested abstract postcondition. Acceptance requires a specified transport of e across θ and a valid target enforcement contract. A successful return supplies $g : k \to k'$ with $k' \in O_\Psi$ and checked evidence of $Q(\psi(k'))$. Failure instead returns a state in the declared failure interface. This typed record distinguishes initial acceptance from eventual completion; it contains no presumed success strategy.

For state predicates, call I *isomorphism-invariant* if $a \cong b$ implies $I(a) \Leftrightarrow I(b)$, and *forward-invariant* if $I(a)$ and $a \to a'$ imply $I(a')$. An evidence functor $E : A \to \mathbf{Set}$ may give witnesses of I when its elements have that declared meaning.

Theorem 10.2 (Invariant transport at migration)**.** *Let a recovered source state o satisfy $I(\phi(o))$, and choose $(o, k, \theta) \in M(\phi, \psi)$. If I is isomorphism-invariant, then $I(\psi(k))$. If it is also forward-invariant, it holds at every later observable target state reachable from k. For a covariant evidence functor E, evidence $e \in E(\phi(o))$ transports to $E(\theta)(e)$ and then along each interpreted target arrow.*

Proof. The first assertion is invariance under θ. For $g : k \to k'$ with $k' \in O_\Psi$, fullness of the domain and soundness give $\psi(g) : \psi(k) \to \psi(k')$. Forward invariance gives the second assertion. The stated evidence maps exist by functoriality of E, and compose by its composition law. □

If only an earlier source checkpoint is known to satisfy I, its preservation through recovery additionally uses forward invariance on the interpreted completed source path. Recovery can advance the task. A claim about the value of an invariant at an uninterpretable intermediate state is not licensed by this theorem.

When migration must be invisible to effect observers, require $\lambda_A(\theta)$ to be an identity, as well as account for the physical migration operation's effects. An arbitrary isomorphism in an action-sensitive category need not be invisible. If only a noninvertible refinement arrow relates the representations, evidence can sometimes still move forward, but reversible state agreement has been replaced by a weaker notion.

Authority does not follow from the matching square. In a common permission space, require a target grant $k_{\text{grant}} \leq c_{\text{grant}}$, a mediated transfer operation, and target confinement

under k_{grant}. If the source credentials remain active, or a target tool can use a larger ambient grant, the intended non-amplification premise fails. Cross-system permission translations need their own verified inequalities.

10.2 What state agreement does not prove

Consider an abstract initial state with two distinct outgoing outcomes. One sound implementation always chooses the left outcome and another always chooses the right. Their initial states match exactly, but their future trace sets differ. Theorem 10.2 preserves an invariant shared by both outcomes; it does not equate their futures.

Forward simulations provide the standard route from related states to matched executions [15]. Here one sufficient extra condition is a *forward checkpoint simulation*. Choose a set R of objects of $M(\phi,\psi)$ such that, for every $(o,k,\theta) \in R$ and every $f : o \to o'$ in O_Φ, there exist $g : k \to k'$ in O_Ψ and $(o',k',\theta') \in R$ satisfying (13). If all comparison isomorphisms have identity action label, applying λ_A to the square equates the interpreted labels of f and g.

Proposition 10.3. *If A carries a label functor λ_A and every comparison isomorphism in R has identity label, such a simulation matches every finite sequence of source checkpoint transitions by a target sequence with equal interpreted labels, starting from each related pair.*

Proof. For the empty sequence use the given related pair. At each successive source arrow, apply the simulation condition at the currently related target state. It supplies the next target arrow and a new related pair. Induction yields the sequence; the label equality follows from each square and functoriality of labels. □

This is finite, existential matching. Mutual simulations give two inclusions of finite checkpoint behaviors from related states. They do not by themselves preserve fairness, termination, timing, or divergence hidden between checkpoints. Transparent migration for a specified client must prove the version of behavioral preservation that the client actually observes. Abadi and Lamport's refinement-mapping framework [16] explicitly treats behaviors and stuttering; a matching checkpoint alone supplies no such run-level result.

10.3 Background controllers and naturality

Control needs a grant
A controller's position in the architecture does not itself confer authority.

A background controller operates on configurations: a computation, its current representation, active grants, and witnesses justifying changes. Its transitions can request recovery, install another implementation, or change a permitted policy. Those transitions require their own authority and evidence. A controller is not more trusted merely because it lies behind the foreground computation.

For a change between two interpretation boundaries, a well-typed compatibility square is

$$\begin{array}{ccc} A & \xrightarrow{H} & A' \\ {\scriptstyle\phi}\uparrow & & \uparrow{\scriptstyle\phi'} \\ O_\Phi & \xrightarrow[F|_{O_\Phi}]{} & O_{\Phi'} \end{array} \tag{14}$$

where $F : C \to C'$ is a functor with $F(O_\Phi) \subseteq O_{\Phi'}$, $H : A \to A'$ is a functor, and the displayed square commutes strictly. It expresses semantic compatibility, not authorization of F or equivalence of H. Partial controllers require an explicit further restriction of the displayed domain.

An ordinary natural transformation has different data: for parallel functors $F, G : X \to Y$, its components $\eta_x : Fx \to Gx$ satisfy

$$\begin{array}{ccc} Fx & \xrightarrow{Ff} & Fx' \\ {\scriptstyle \eta_x}\downarrow & & \downarrow{\scriptstyle \eta_{x'}} \\ Gx & \xrightarrow[Gf]{} & Gx' \end{array} \tag{15}$$

for every $f : x \to x'$ in X. The components are arrows in Y. A heterogeneous migration between states of unrelated categories is not automatically such a transformation. If transformations are proposed uniformly across a tower, naturality within levels and compatibility with the interpretation squares must both be checked.

Instrumentation often supplies a forgetful functor $U : \widetilde{C} \to C$. For example, let $C = B\mathbb{N}$ record the number of loop iterations as arrows at one state. Let $\widetilde{C}$ have objects $n \in \mathbb{N}$ and a unique arrow $n \to n + r$ for each $r \geq 0$. Define $U(n \to n + r) = r$. This is a functor erasing the explicit counter. It has no functorial section: a section would choose an object n, and the generator of $B\mathbb{N}$ would have to map to an endomorphism of n whose image is 1; the only such endomorphism is the identity, whose image is 0. Thus adding this instrumentation is not an inverse functor, much less automatically a natural transformation. Logging also needs separate confidentiality, cost, and failure analyses.

11 Worked operational examples

The following are explicit mathematical models and conditional engineering interpretations. They are not claims about a deployed payment service or a certified physical controller.

11.1 Authority-constrained payments and agent migration

WORKED EXAMPLE / DELEGATION AND MIGRATION

Take a task to pay approved invoices to a fixed merchant, subject to a total budget $B \in \mathbb{N}$. The permission set distinguishes proposing an invoice, approving a named invoice, executing a payment, and changing policy. An agent allowed to propose does not receive the execution permission. A receiver's representation of an approved invoice includes its identifier, amount, merchant, approval evidence, and a one-use settlement identifier.

Reusable permission to execute is insufficient to bound total expenditure. Introduce a ledger state $(s, r_1, \ldots, r_n, N)$, where s is the amount already settled, r_i is agent i's residual allocation, and N is the set of consumed settlement identifiers. The model maintains

$$s + \sum_i r_i \leq B, \qquad s, r_i \geq 0. \tag{16}$$

An atomic payment of amount m by agent i requires a valid invoice approval, the execution permission, $0 \leq m \leq r_i$, and a fresh identifier $\nu \notin N$. Its transition is

$$(s, r_i, N) \longmapsto (s + m, r_i - m, N \cup \{\nu\}),$$

leaving other residual allocations unchanged. A delegation split replaces r_i by $r_i - d$ and creates a new allocation d, with $0 \leq d \leq r_i$. The recipient receives only attenuated permissions scoped to the task.

Proposition 11.1. *Starting from a state satisfying* (16)*, any finite sequence of these payment and split transitions preserves* (16)*; no settlement identifier can be paid twice.*

Proof. A payment adds m to s and subtracts the same amount from one residual, preserving their sum and nonnegativity. A split also preserves the sum and nonnegativity.

Payment boundary

The ledger proves a budget bound. Atomic updates, fresh identifiers, and external deduplication are explicit premises.

WORKED EXAMPLE / DELEGATION AND MIGRATION (continued)

Induction proves the budget invariant. The freshness precondition and monotone growth of N exclude reusing an identifier. □

For $B = 100$, allocations 60 and 40 satisfy the invariant. Two independently copied grants of 100 do not. Concurrent payments are justified only if the ledger transition and freshness test are atomic, or if a proved protocol implements their abstract atomic behavior. A linearizability proof [17] can justify this passage from concurrent histories to the atomic ledger, provided the specification includes the entire budget-and-freshness update. A database transaction, capability token, or signed message is not by itself a proof of that property.

Now migrate the second agent after it has prepared a payment but before receiving a settlement response. The abstract task state must distinguish unsubmitted, pending, and confirmed settlements. A migration that copies only the residual budget can lose the pending identifier and resend the request, invalidating the model. A valid matching witness preserves the pending request, identifier, residual allocation, and relevant approval version. The old agent must lose the ability to commit a competing continuation, or the ledger must reject it through an epoch or deduplication rule represented in the model.

Theorem 10.2 then transports the budget invariant and its evidence. Target enforcement under the attenuated grant supplies authority preservation. End-to-end prevention of duplicate external settlement additionally assumes or proves that the payment endpoint honors the same identifier discipline. This is a concrete instance of why state transport, authority transfer, and interaction with the environment require separate witnesses.

11.2 A transactional industrial interlock

WORKED EXAMPLE / INDUSTRIAL FORMALISATION

Physical boundary

The invariant concerns commanded registers. Physical interlock safety needs an additional model and refinement.

Consider a controller whose commanded output is a pair $a = (d, h) \in \{0, 1\}^2$, where $d = 1$ commands a closed guard and $h = 1$ enables a heater. Let

$$V = \{(d, h) \mid h \leq d\}.$$

The abstract category A is the free path category of the graph with vertex set V and an update edge $a \to b$ for each permitted command update. For this example all pairs in $V \times V$ are permitted. Labels record the sequence of committed target commands. The invariant $h \leq d$ holds at every abstract object.

A concrete controller has a live register $a \in V$, a shadow register $b \in \{0, 1\}^2$, and a phase. In idle phase $b = a$. A begin step enters preparation. Preparation steps edit shadow bits; a commit is allowed only when $b \in V$ and atomically sets the live register to b and returns to idle. An abort discards the shadow and returns to idle with the live register unchanged. Let C be the free category of these microsteps. Observable states are the idle states, forming a full subcategory O.

Interpret an observable state by its live register and an observable path by its sequence of commits. Begin, shadow edits, and aborts contribute no abstract edge. Each recorded commit has exactly the source live register and target checked shadow register required in A, so the interpretation is well defined. Concatenation concatenates commit sequences; the empty path maps to an identity. Thus it is a functor, with compatible command labels. Every abstract state has an idle realization. Every abstract update lifts from its prescribed idle realization by beginning, editing the shadow to the target, and committing. Therefore the interpretation is representation-total and complete. It is observable using the safe subcategory generated by aborts and identities: an idle state uses its identity, and a

WORKED EXAMPLE / INDUSTRIAL FORMALISATION (continued)

preparation state can abort. A source execution interrupted in preparation can be inspected after abort without requiring a new external input.

This is also an example where safe recovery and semantic progress differ. Recovery discards uncommitted work and contributes no committed command; another implementation could instead safely complete a pending command. Either choice must be stated in the contract. Here the visible invariant holds at *all* concrete states because only a validated atomic commit changes the live register. It is proved directly from the microstep rules, rather than inferred solely from endpoint soundness.

For authority, grant one permission to update the commanded output. Assign that requirement to each commit and to each abstract update; internal shadow operations and aborts require none in this model. Observable path requirements are exactly the joins of their commit requirements, so (AP) holds with identity translation. Adding a privileged maintenance operation that bypasses validation would require changing the category or rejecting the authority-preservation claim.

This model describes commanded values. It does not prove that a physical guard closes instantaneously, that sensors report correctly, or that a heater cools before access becomes possible. Those facts require a physical-state model, environmental assumptions, and an additional checked interpretation. Even atomicity of the live-register update is a hardware or lower-level refinement obligation. The need to prove each implementation boundary is exemplified by verified compilation [18]; the model here supplies no comparable hardware-to-physics theorem. The example separates a proved finite protocol property from the stronger industrial assurance argument it could support.

12 Which additional categorical constructions help?

The elementary core already gives composition, recovery, and state transport. Additional constructions are valuable when they express additional data, not when they merely rename an informal architecture.

Structure must do work

Universal liftings, relational actions, and two-dimensional pasting are additional mathematical data.

Fibrations and context changes. Fibred categories make variation over a base category explicit [6, 19]. Suppose theories really admit contravariant reindexing $F : D^{\mathrm{op}} \to \mathbf{Cat}$. Its Grothendieck construction has objects (d, x) with $x \in F(d)$ and arrows $(d', x') \to (d, x)$ given by $u : d' \to d$ and $x' \to F(u)(x)$. The projection is a fibration. Concretely, the arrow $(d', F(u)x) \to (d, x)$ above u is cartesian: every arrow into (d, x) whose base factors through u factors uniquely through this lift over the prescribed remaining base arrow. This universal factorization, not a metaphor about context, is the content of the construction.

If each context has an interpretation, reindexing must preserve its domains and satisfy a square such as (17). Here u^*_O is the restriction of concrete reindexing to the observable domains.

$$\begin{array}{ccc} O_d & \xrightarrow{\phi_d} & A_d \\ {\scriptstyle u^*_O}\downarrow & & \downarrow{\scriptstyle u^*_A} \\ O_{d'} & \xrightarrow[\phi_{d'}]{} & A_{d'} \end{array} \tag{17}$$

The equality is an additional compatibility condition. A change of policy, jurisdiction, or sensor assumptions need not admit such a reindexing at all. When it does not, new proof obligations should be exposed. Calling the square Beck–Chevalley does not supply adjoints or a base-change theorem; those would be further structure. For covariant evidence transport, the discrete *opfibration* of Proposition 6.2 has the opposite lifting direction.

Profunctors and nonunique realizations. A profunctor $P : X^{\mathrm{op}} \times Y \to \mathbf{Set}$ supplies sets of generalized correspondences together with coherent pre- and post-composition actions; their role as arrows between categories is developed by Grandis and Paré [20]. Given $\phi : O \to A$, the formula $P(o, a) = A(\phi(o), a)$ is such a profunctor. It records a representation together with a residual abstract arrow; it is not simply the inverse-image set of a. A bare relation "o represents a" does not generally have the actions required of a profunctor. Profunctors become useful for genuinely relational compilation or open interfaces, but their composition and preservation properties would require a separate development. Our source-lifting and iso-comma constructions avoid this overhead for the stated results.

Double categories and control squares. Using the double-category conventions of Grandis and Paré [20, Section 1], there is an elementary strict double category with categories as objects, functors as both horizontal and vertical arrows, and strictly commuting functor squares as cells. Horizontal and vertical pasting are defined by composition; identity squares are evident, and the interchange equation follows by associativity of functor composition. This accurately packages squares such as (14) after restricting to the displayed observable domains.

It does not validate a diagram whose objects are individual task states but whose vertical arrows are functors between whole categories. Nor can a simulation relation, a natural transformation, and a strict commuting square be substituted for one another without changing the cell structure and proving the new composition laws. A useful double category of partial interpretations and authorized reconfigurations must additionally control domains under both kinds of pasting. That construction is a research direction, not a premise of the present confinement theorem.

13 Limits and a concrete research programme

Proof status

The results have explicit mathematical proofs. Mechanization and deployment validation remain separate work.

The established results are deliberately conditional and finite in scope. They give a category of partial interpretations, explicit realization properties, a sufficient recovery-composition theorem, evidence transport, a permission-confinement theorem, and invariant-preserving migration. They do not establish universal liveness, full abstraction for arbitrary observers, or physical security. The worked examples demonstrate that the same definitions apply to proof artefacts and agent tasks, while making their additional assumptions visible.

Three developments are particularly consequential. First, a mechanized library should encode full-domain partiality, safe lifts, permission requirements, and the exact hypotheses of Theorems 4.4, 8.5, and 10.2. It should include their small counterexamples as regressions. Second, a run-sensitive extension should specify adversarial scheduling, divergence, deadlines, and distributed recovery, so that migration can preserve the observations clients actually depend on. Third, proof-inspection tooling should attach verified obligation maps and conservative assumption manifests to real kernel artefacts, then test whether reviewers can identify specification mistakes and authority changes more reliably.

These tasks have different validation criteria. Mechanization checks formal derivations; model validation checks correspondence with an implementation and its environment; user studies assess whether inspection is effective. None can substitute for the others. Semantic towers provide a precise place to attach these obligations and compose the guarantees that have actually been established.

References

[1] F.-R. Rideau. “Climbing Up the Semantic Tower—at Runtime”. Position paper presented at Off the Beaten Track, POPL 2018. 2018. URL: https://popl18.sigplan.org/details/OBT-2018/12/Climbing-Up-the-Semantic-Tower-at-Runtime.

[2] S. Mac Lane. *Categories for the Working Mathematician*. 2nd ed. Graduate Texts in Mathematics 5. Springer, 1998. DOI: 10.1007/978-1-4757-4721-8.

[3] J. R. B. Cockett and S. Lack. “Restriction Categories I: Categories of Partial Maps”. In: *Theoretical Computer Science* 270.1-2 (2002), pp. 223–259. DOI: 10.1016/S0304-3975(00)00382-0.

[4] B. Alpern and F. B. Schneider. “Defining Liveness”. In: *Information Processing Letters* 21.4 (1985), pp. 181–185. DOI: 10.1016/0020-0190(85)90056-0.

[5] G. C. Necula. “Proof-Carrying Code”. In: *24th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL)*. ACM, 1997, pp. 106–119. DOI: 10.1145/263699.263712.

[6] B. Jacobs. *Categorical Logic and Type Theory*. Studies in Logic and the Foundations of Mathematics 141. North-Holland, 1999. URL: https://www.cs.ru.nl/B.Jacobs/CLT/bookinfo.html.

[7] M. J. C. Gordon, R. Milner, and C. P. Wadsworth. *Edinburgh LCF: A Mechanized Logic of Computation*. Lecture Notes in Computer Science 78. Springer, 1979. DOI: 10.1007/3-540-09724-4.

[8] C. A. R. Hoare. “An Axiomatic Basis for Computer Programming”. In: *Communications of the ACM* 12.10 (1969), pp. 576–580, 583. DOI: 10.1145/363235.363259.

[9] A. W. Appel and E. W. Felten. “Proof-Carrying Authentication”. In: *6th ACM Conference on Computer and Communications Security (CCS)*. ACM, 1999, pp. 52–62. DOI: 10.1145/319709.319718.

[10] J. C. Reynolds. “Separation Logic: A Logic for Shared Mutable Data Structures”. In: *17th IEEE Symposium on Logic in Computer Science (LICS)*. IEEE, 2002, pp. 55–74. DOI: 10.1109/LICS.2002.1029817.

[11] M. S. Miller. “Robust Composition: Towards a Unified Approach to Access Control and Concurrency Control”. PhD thesis. Johns Hopkins University, 2006. URL: https://papers.agoric.com/assets/pdf/papers/robust-composition.pdf.

[12] J. H. Saltzer and M. D. Schroeder. “The Protection of Information in Computer Systems”. In: *Proceedings of the IEEE* 63.9 (1975), pp. 1278–1308. DOI: 10.1109/PROC.1975.9939.

[13] R. Milner. “Fully Abstract Models of Typed λ-Calculi”. In: *Theoretical Computer Science* 4.1 (1977), pp. 1–22. DOI: 10.1016/0304-3975(77)90053-6.

[14] J. A. Goguen and J. Meseguer. “Security Policies and Security Models”. In: *IEEE Symposium on Security and Privacy*. IEEE, 1982, pp. 11–20. DOI: 10.1109/SP.1982.10014.

[15] N. Lynch and F. Vaandrager. “Forward and Backward Simulations: I. Untimed Systems”. In: *Information and Computation* 121.2 (1995), pp. 214–233. DOI: 10.1006/inco.1995.1134.

[16] M. Abadi and L. Lamport. “The Existence of Refinement Mappings”. In: *Theoretical Computer Science* 82.2 (1991), pp. 253–284. DOI: 10.1016/0304-3975(91)90224-P.

[17] M. P. Herlihy and J. M. Wing. “Linearizability: A Correctness Condition for Concurrent Objects”. In: *ACM Transactions on Programming Languages and Systems* 12.3 (1990), pp. 463–492. DOI: 10.1145/78969.78972.

[18] X. Leroy. “Formal Verification of a Realistic Compiler”. In: *Communications of the ACM* 52.7 (2009), pp. 107–115. DOI: 10.1145/1538788.1538814.

[19] J. Bénabou. “Fibered Categories and the Foundations of Naive Category Theory”. In: *The Journal of Symbolic Logic* 50.1 (1985), pp. 10–37. DOI: 10.2307/2273784.

[20] M. Grandis and R. Paré. “Limits in Double Categories”. In: *Cahiers de Topologie et Géométrie Différentielle Catégoriques* 40.3 (1999), pp. 162–220. URL: https://archive.numdam.org/item/CTGDC_1999__40_3_162_0/.