Coherence Types

A Refinement Calculus where Convention Coherence is the Declarations Forcing Every Reconciliation — Parts I (Statics) and II (Dynamics and Soundness)

John Komkov, Independent Researcher

June 2026

Contents

Abstract

Hoare's rule for sequential composition is sound only when the predicate at the seam has a stable meaning on both sides. Open agent/tool compositions break that assumption: each component satisfies its local contract, but the conventions the components attach to shared quantities — date bases, unit scales, currencies, timezones — are transported across interfaces and need not agree where a feedback loop closes. A companion program identifies the forced obstruction invariant — the coherence fee, the witness rank of a semantic interface complex (Komkov, A Witness Logic for Semantic Composition, 2026) — and a second companion proves it cannot be recovered from local observation (Komkov, Compositional Incoherence is a Parity, 2026: SQ-hard and LPN-hard). The negative result has a constructive corollary: if coherence cannot be inferred bottom-up, it must be carried top-down, as a typing invariant. This paper builds that type system, in two parts. Part I (statics) presents λ_∇, a refinement calculus for tool composition: types carry convention bundles (per-dimension frames); a declared tool posts a known transport between frames while an opaque tool emits a value of latent (undeclared) convention; reconciliation posts the constraint that two frames agree; and disclosure declares a latent frame. The typing context, read as a presheaf over the composition poset, is a semantic interface complex. Coherence decomposes into two conditions that must not be conflated: consistency (the declared constraints and reconciliations have a common solution — no clash) and `fee = 0` (every reconciliation is forced by the declarations — no undeclared convention is load-bearing), where `fee = rank(O ∪ R) − rank(O)` for `O` the declared constraints and `R` the reconciliations —

the companion's witness rank, computed as the deployed checker computes it. A coercion is an admissible disclosure; the minimum coercion set is the disclosure normal form, of size `fee`. Part II (dynamics and soundness) gives a tagged operational semantics in which a `reconcile` whose two runtime frames disagree is convention-stuck, and proves Coherence Soundness: a coherent program enjoys progress — it is a value or steps, and never reaches a convention-stuck state, for every behavior of the opaque tools — and preservation — reduction preserves typing and coherence. The progress proof is the payoff and it is short once the statics are right: at runtime the declared constraints `0` hold by construction, and `fee = 0` makes every reconciliation a linear consequence of `0`, so consistency forces it to hold. The calculus is Hindley–Milner unification with a cohomological twist — a cycle of frame constraints is satisfiable iff its holonomy is trivial — and its soundness theorem is the positive half of the program's thesis: coherence is a global typing invariant precisely because it is locally unlearnable.

---

1 1. Introduction

1.1 1.1 The missing composition rule, as a typing problem

Hoare's sequential-composition rule

$$\frac{\{P\}\, C_1\, \{R\} \qquad \{R\}\, C_2\, \{Q\}}{\{P\}\, C_1; C_2\, \{Q\}}$$

is sound because the assertion `R` means the same thing where `C₁` leaves off and where `C₂` picks up. Open compositions violate this silently: a calendar tool emits a date in ISO-8601 with implicit UTC; an invoicing tool reads it in implicit local time; each interface is type-correct, each local contract holds, yet the composition is wrong by a timezone offset, invisible to either tool's local audit ([CD] §1; provably invisible to any bounded-locality observation, [PAR]). The seam moved the semantic frame — the convention under which a value is read.

The companion program names the obstruction. A composition is a semantic interface complex ([CD] Def 2.1): a presheaf `F` of conventions over a poset of contexts, an observable subpresheaf `0 ⊆ F` (declared conventions), and a latent quotient `L = F/0` (undeclared ones). The coherence fee is the witness rank, `[CD]`'s unique invariant respecting the primitive of repair (disclosure). This paper makes the apparatus a type system: the missing composition rule is the missing typing rule — the side condition that a composition's conventions glue.

1.2 1.2 The thesis, stated precisely

We build λ_∇ ("lambda-connection"). A term generates, per dimension `d`, a system of group-valued constraints on the frames (conventions) of its value occurrences:

- a declared tool posts a known relation `x_out = g + x_in` (its declared transport);

- an opaque tool emits a value whose frame is latent — a fresh free variable, posting no relation (the tool does not declare its convention);
- a reconciliation `reconcile_d(v₁, v₂)` posts the constraint `x_{v₁} = x_{v₂}` — the seam, the only former (in the recursion-free core) that demands two independently-derived frames agree;
- a disclosure `disclose_d(v, φ)` posts a pin `x_v = φ`, declaring a latent frame.

Write `O` for the declared constraints (declared transports and pins) and `R` for the reconciliation constraints. The central definitions:

> Coherence `Γ ⊢ e ✓` :⟺ `O ∪ R` is consistent (no clash) and `fee(e) = 0`, where `fee(e) := rank(O ∪ R) − rank(O)` is the number of reconciliations not forced by the declarations.

Two failure modes, genuinely different and not to be conflated:

- Inconsistency (a clash): `O ∪ R` has no solution — the declared transports and a reconciliation contradict (a settlement loop whose declared arithmetic does not compose; or two pins reconciled at mismatched frames). A hard type error; `fee` (a dimension) does not see it.
- Under-determination (positive fee): `O ∪ R` is solvable but some reconciliation is not a consequence of `O` — it constrains a latent frame the declarations leave free, so at run time that frame (set by an opaque tool) can violate it. Repairable by disclosure.

The dictionary, each entry made precise in §2–§3:

Type-theoretic notion	Companion object
A type's convention bundle	A stalk of the convention presheaf `F` ([CD] Def 2.1)
Typing context `Γ`, as a presheaf over the composition poset	A semantic interface complex `G_e ∈ SemComp`
Declared transport / opaque tool	Observable restriction map / a latent stalk in `L = F/O`
Coherence	Consistency and `fee = 0`
Convention clash (hard error)	Inconsistent declared constraints (`O ∪ R` unsatisfiable)
Coherence fee	Witness rank `rank(O ∪ R) − rank(O) = dim H¹` ([CD])
A coercion	An admissible disclosure ([CD] §2.3 III)
Minimum coercion set	The disclosure normal form ([CD] Thm 7.1), size `fee`
Convention-mismatch stuck state (Part II)	Holonomy `≠ 0` realized at run time

The one-line summary: λ_∇ is Hindley–Milner unification with a cohomological twist. In ordinary HM a cycle of equality constraints `α = β = ⋯ = α` is always satisfiable. In λ_∇ the constraints carry group elements, and a cycle is satisfiable iff its holonomy (the signed sum of transports) is `0`; a reconciliation that closes such a cycle is forced only when the declarations fix that holonomy. This is the exact sense in which the separation-logic frame rule fails ([CD] §6.3 calls the fee "the minimum disclosure required to recover frameability").

1.3 1.3 Why a type system: the bridge from the hardness result

[PAR] proves the obstruction is a parity, hence (i) invisible to bounded-locality observation and (ii) hard to recover from noisy observation under LPN/LWE. The constructive contrapositive is this paper's reason to exist:

> Coherence cannot be recovered bottom-up from observation; therefore it must be carried top-down, as a typing obligation declared at the interface.

λ_∇ is that declared structure; the certificate is the typing derivation. B2 (negative) and B3 (positive) are a matched pair — coherence is a global typing invariant precisely because it is locally unlearnable.

1.4 1.4 Contributions

Part I (§2–§3). The calculus λ_∇ (recursion-free core; §2); its convention system (`0`, `R`); the constraint-generation statics (§3.1); the consistency/fee decomposition and the coherence judgment (§3.2–3.3); decidability of coherence as a polynomial-time rank (§3.2); the exact-regime conditions making the fee the companion's witness rank (§3.3); coercion = admissible disclosure and minimum coercion = disclosure normal form (§3.4); frame polymorphism (§3.5).

Part II (§4–§5). A tagged operational semantics with the convention-stuck state (§4); the main result, Coherence Soundness — progress (`coherent` ⇒ value-or-steps, never convention-stuck, for every opaque behavior) and preservation (§5) — with the elaboration-soundness corollary (elaboration inserts `fee` coercions and yields a sound program).

Deferred: the graded disclosure modality `□_r` (grade = fee), general feedback/recursion, and interface-evolution-as-preservation ([CD] Conj 9.5) are Part III; mechanization and the Bulla elaborator are Part IV.

1.5 1.5 Claim discipline and scope

[Established, cited]: the semantic-interface-complex model, the witness rank and its TU incidence-rank computation (Bulla), repair duality (Thm 6.2), the disclosure normal form (Thm 7.1), the DFD+CHP⇒exact sufficiency — from [CD] and the hierarchical-fee companion [HF]; the holonomy/flatness classification from [LV]; the hardness motivation from [PAR]. [New,

proven here]: λ_∇ and its statics; the consistency/fee decomposition and coherence characterization (Prop 3.5); the exact-regime conditions for λ_∇'s complexes (Prop 3.3); coercion = disclosure (Prop 3.6); decidability (Prop 3.2); and the Coherence Soundness theorem (Thm 5.4: progress + preservation). Scope: the abelian, exact, recursion-free regime — each dimension has an abelian additive structure group; constraints are single-dimension and rank-1; declared transports are fixed (not frame-dependent); the core has no recursion (reconciliation is the sole loop-closer). Frame-dependent transports, cross-dimension coupling, multi-frame interfaces (obstruction in H^2/gerbes, [SCPI]), and general feedback are the frontier, named and out of scope (§3.7).

---

2 2. The calculus λ_∇

2.1 2.1 Frames, declared transports, and opacity

Fix a finite set `D` of convention dimensions. For each `d` fix an abelian group $\mathbb{E}$_d, written additively, the structure group, acting freely and transitively on a nonempty set $\mathbb{F}$_d of frames: two frames differ by a unique transport `ψ − φ ∈ 𝔼_d`. (Multiplicative groups such as unit-scaling are presented additively via a fixed logarithm; we restrict to groups admitting such a presentation, so the incidence computation of §3.2 is over `ℚ` and totally unimodular.)

A tool transforms a value. On each relevant dimension `d` it is either:

- declared: it posts a known transport `g^t_d ∈ 𝔼_d`, so its output frame is `g^t_d +` its input frame (a known relation between the two frames); or
- opaque: it does not declare its convention behavior, so its output frame is a latent frame — a fresh free variable, related to nothing. The calendar tool emitting "implicit UTC" is opaque on `timezone`: it produces a date whose frame is undeclared.

Latent frames are the hidden data the fee measures; a latent frame becomes observable only by disclosure. This is [CD]'s `O ⊆ F` split: declared frames generate the observable `O`, latent ones the quotient `L = F/O`. Note transports of declared tools are known and fixed — coherence is not about discovering them but about whether the declared structure forces the reconciliations.

2.2 2.2 Types

Types carry, on relevant dimensions, a frame term `φ` — a concrete frame or a frame variable `α`:

$$\tau ::= \mathsf{b}\langle d : \varphi\rangle \mid \tau_1 \to \tau_2 \mid \tau_1 \times \tau_2 \mid \mathsf{unit}, \qquad \varphi ::= \phi \in \mathbb{F}_d \mid \alpha.$$

`b⟨d:φ⟩` is a single-dimension interface type (one relevant `d` per base value; a value relevant to several dimensions is a product). The single-dimension restriction makes the per-dimension constraint systems independent (Prop 3.3); multi-frame interfaces, where dimensions couple,

are the non-abelian frontier (§3.7). A frame variable is observable if a derivation pins it, latent otherwise.

2.3 2.3 Terms

λ_∇ is the simply-typed λ-calculus — without recursion or fixpoint (§3.7) — plus tools, a binder, reconciliation, and disclosure:

$$
\begin{aligned}
e ::= {} & x \mid () \mid (e_1, e_2) \mid \pi_i\, e \mid \lambda x{:}\tau.\, e \mid e_1\, e_2 \mid \mathsf{let}\ x{=}e_1\ \mathsf{in}\ e_2 \\
& \mid t(e) \quad \text{(tool: declared posts a transport, opaque posts a latent frame)} \\
& \mid \mathsf{reconcile}_d(e_1, e_2) \quad \text{(seam: assert the two frames agree)} \\
& \mid \mathsf{disclose}_d(e, \phi) \quad \text{(coercion: declare } e\text{'s } d\text{-frame is } \phi).
\end{aligned}
$$

`reconcile_d(e₁,e₂)` returns the common value and posts `x_{e₁} = x_{e₂}` — the only former of the recursion-free core that can demand two independently-built frames agree (Prop 2.2). `disclose_d(e,φ)` pins `e`'s frame to `φ`; in elaboration (§3.4) disclosures are inserted automatically.

2.4 2.4 The convention system of a term; the context as presheaf

A well-scoped term `e` generates, per dimension `d`, a finite set of frame variables (one per `d`-relevant value occurrence) and two sets of group-valued affine constraints over them:

- the declared constraints `O_{e,d}`: one `x_v = g^t_d + x_u` per declared tool edge `u→v`, and one pin `x_v = φ` per disclosure;
- the reconciliation constraints `R_{e,d}`: one `x_{v_1} = x_{v_2}` per `reconcile_d`.

Opaque tools post no constraint (their output is a fresh free variable). Collecting these over `d` gives the convention sheaf `G_e ∈ SemComp` ([CD] Def 2.1): contexts are value occurrences ordered by dataflow; the stalk at a node is its frame; declared edges are restriction maps; the observable subpresheaf `O_e` is generated by the declared constraints; the latent `L_e = F_e/O_e` is carried by the opaque outputs. The typing context `Γ`, read as a presheaf over the composition poset, is `G_e`.

Definition 2.1 (holonomy). For a cycle in the graph of declared + reconciliation edges, its holonomy is the signed sum of transports along it (reconciliation edges contribute `0`). A cycle is coherent iff its holonomy is `0`. (This is [LV] Def 2.3, additively.)

Proposition 2.2 (reconciliation is the sole loop-closer). In the recursion-free core, the dataflow is a DAG, so the declared / opaque tool edges form no cycle; every cycle in `G_e` uses a reconciliation edge or two pins meeting at a pinned frame.

Proof. Without `fix`, `let` / application build a finite DAG of value dependencies; tool edges follow it and form no cycle. The only edges against the DAG are reconciliation edges and pins. ■

2.5 2.5 Two running examples

(a) Settlement — a clash. The pipeline of [LV] Ex 2.9, with `risk` and `treasury` declared on the `settlement-instant` dimension (known day-arithmetic transports `g_r, g_t`):

```
let cap = capture()            -- cap : Date⟨d:α⟩            (α a frame variable)
let h   = risk(cap)            -- declared: posts  x_h = g_r + x_cap
let rel = treasury(h)          -- declared: posts  x_rel = g_t + x_h
in  reconcile_d(cap, rel)      -- R: posts  x_cap = x_rel
```

`O` forces `x_rel − x_cap = g_t + g_r`; `R` demands `x_rel − x_cap = 0`. So `R` is a consequence of `O` (`fee = 0`), but `O ∪ R` is inconsistent unless `g_t + g_r = 0`. The day-arithmetic is off by one banking day, so `g_t + g_r ≠ 0`: a clash, a hard type error — the declared transports genuinely do not compose. (No disclosure repairs a clash; one must fix the arithmetic. [LV] §2.9 instead treats the basis as frame-dependent — the transport depends on the undeclared basis — which is the cross-dependency frontier of §3.7; there the same situation is `fee = 1`. In the scoped fixed-transport model it is a clash, and we report it as one.)

(b) An undeclared convention — `fee = 1`. A genuinely under-determined reconciliation:

```
let a = fetch_rate()           -- declared: a : Rate⟨d:φ_usd⟩  (pinned, observable)
let b = legacy_quote()             -- OPAQUE on d: b : Rate⟨d:β⟩     (β latent —
legacy tool, undeclared scale)
in  reconcile_d(a, b)          -- R: posts  x_a = x_b
```

`O = {x_a = φ_usd}`; `R = {x_a = x_b}` with `x_b` a latent free variable (opaque output). `O ∪ R` is consistent (set `x_b = φ_usd`), but `R` is not a consequence of `O` (it constrains the free `x_b`): `fee = rank(O ∪ R) − rank(O) = 2 − 1 = 1`. The legacy quote's scale is undeclared, so the type-checker cannot certify the rates agree. One coercion repairs it — `disclose_d(b, φ_usd)` declares the scale, pinning `x_b`; then `R` follows from `O`, `fee = 0`, and the term is coherent. One obstruction, one coercion: `fee = ρ = 1`, repair duality ([CD] Thm 6.2).

---

3 3. Statics

3.1 3.1 Constraint generation

`Γ ⊢ e : τ ⇝ (O; R)` emits the declared (`O`) and reconciliation (`R`) constraints, under a freshness discipline (each value occurrence draws frame variables from a global counter, so `(O;R)` is canonical). The structural rules (`Var`, `Unit`, `Pair`, `Proj`, `Abs`, `App`, `Let`) thread frame variables unchanged and emit nothing; `App` matches the argument's frame terms to the parameter's, emitting only forward identity constraints into `O` (no back-edge: by Prop 2.2 the argument is built earlier in the DAG). `Let` generalizes (frame variables and transports) via `Gen_Γ` (§3.5). The characteristic rules:

$$\frac{\Gamma \vdash e : \mathsf{b}\langle d{:}\alpha\rangle \rightsquigarrow (O; R) \quad t \text{ declared, } g_d^t}{\Gamma \vdash t(e) : \mathsf{b}'\langle d{:}\beta\rangle \rightsquigarrow (O \cup \{\beta = g_d^t + \alpha\};\ R),\ \beta \text{ fresh}} \text{ (Tool-decl)}$$

$$\frac{\Gamma \vdash e : \mathsf{b}\langle d{:}\alpha\rangle \rightsquigarrow (O; R) \quad t \text{ opaque on } d}{\Gamma \vdash t(e) : \mathsf{b}'\langle d{:}\beta\rangle \rightsquigarrow (O;\ R),\ \beta \text{ fresh (latent — no constraint)}} \text{ (Tool-opaque)}$$

$$\frac{\Gamma \vdash e_1 : \mathsf{b}\langle d{:}\alpha\rangle \rightsquigarrow (O_1; R_1) \quad \Gamma \vdash e_2 : \mathsf{b}\langle d{:}\beta\rangle \rightsquigarrow (O_2; R_2)}{\Gamma \vdash \mathsf{reconcile}_d(e_1, e_2) : \mathsf{b}\langle d{:}\alpha\rangle \rightsquigarrow (O_1 \cup O_2;\ R_1 \cup R_2 \cup \{\alpha = \beta\})} \text{ (Recon)}$$

$$\frac{\Gamma \vdash e : \mathsf{b}\langle d{:}\alpha\rangle \rightsquigarrow (O; R)}{\Gamma \vdash \mathsf{disclose}_d(e, \phi) : \mathsf{b}\langle d{:}\phi\rangle \rightsquigarrow (O \cup \{\alpha = \phi\};\ R)} \text{ (Disclose)}$$

(Tool-decl) adds a declared relation to `O`; (Tool-opaque) adds nothing (latent output); (Recon) adds the seam constraint to `R`; (Disclose) adds a pin to `O`. Generation is syntax-directed, linear in `|e|`.

3.2 3.2 Consistency, the fee, and decidability

Treat `O` and `R` as affine systems `M_O x = b_O`, `M_R x = b_R` over `Q^{D}` (one block per dimension; the rows of `M_O` are declared transports and pins, of `M_R` the reconciliations; entries are `{-1,0,+1}` signed-incidence scaled by transports, with offsets `b`).

Definition 3.1 (consistency; fee). `e` is consistent iff the combined system `O ∪ R` has a solution. The coherence fee is

$$fee(e) := \text{rank} \begin{bmatrix} M_O \\ M_R \end{bmatrix} - \text{rank}(M_O),$$

the number of reconciliation constraints linearly independent of the declared constraints — equivalently `dim(R \bmod \operatorname{span} O)`, the reconciliations not forced by the declarations. This is `[CD]`'s witness rank `rank(δ_F) − rank(δ_O)` under the identification `O = δ_O` (observable), `O ∪ R`-row- space `= δ_F` (full). It is the witness rank the deployed checker reports ([CD] §10.1); that checker computes it via the Kron-reduced Gram, which equals this incidence-rank difference in the exact regime ([CD] §3.5). `λ_∇` uses only the elementary incidence form (§3.3).

Consistency is a value condition (do the declarations and reconciliations have a common solution?); the fee is a structural dimension (how many reconciliations are not forced?). They are independent: a clash can occur at `fee = 0` (settlement: `R ⊆ span O` but the RHS contradicts — §2.5a), and `fee > 0` can occur with no clash (an undeclared convention, satisfiable but not forced — §2.5b). A term whose every tool is declared on every relevant dimension has `fee ≡ 0` (no latent frames), so the fee machinery (§3.4) fires exactly on the opaque tools that model undeclared conventions.

Proposition 3.2 (decidability). Consistency and `fee(e)` are decidable in time `O(|e|^ω)` over `Q`.

Proof. Both are rank computations on the `{−1,0,+1}`-incidence-with-transport matrices `M_O`, `[M_O;M_R]`: `fee` is the rank difference; consistency is Rouché–Capelli (`rank[M_{O∪R}] = rank` of its augmented matrix `[M_{O∪R}\,|\,b]`). Gaussian elimination over `Q` is polynomial; the unweighted incidence pattern is totally unimodular, so the fee rank is field-independent (matching [CD]'s signed-incidence TU, [SI]). ■

Proposition 3.3 (λ_∇ complexes are exact; the fee is the witness rank). Under the core's discipline — single-dimension interface types, one frame per node, per-dimension constraints — the per-dimension systems are mutually independent and rank-1, so `G_e` satisfies disjoint field decomposition and the class-homogeneous partition of [HF]; hence `G_e` lies in [CD]'s exact regime and `fee(e)` of Def 3.1 equals the witness rank `dim H¹_seam(G_e)`.

Proof. Every rule of §3.1 names a single dimension and the single-dimension interface forbids a node bearing two frames, so the constraint set partitions across dimensions with no coupling (DFD), each node carrying one frame (CHP). By [HF] (Theorem 2: DFD+CHP ⇒ exact regime — proved there, not in [CD]), the seam complex truncates to two terms and the witness rank equals the two-term incidence-rank formula, which is exactly `rank[M_O;M_R] − rank M_O` ([CD] §3.5, the Bulla computation). ■

We make no claim about [CD]'s Kron/Schur identity (Lemma 3.9); λ_∇ uses the elementary incidence-rank form, which is all the checker needs.

3.3 3.3 The coherence judgment

Definition 3.4 (coherence). `Γ ⊢ e : τ ✓` :⟺ `Γ ⊢ e : τ ⇝ (O;R)`, `O ∪ R` is consistent, and `fee(e) = 0`.

Proposition 3.5 (coherence = the declarations force every reconciliation). `Γ ⊢ e : τ ✓` iff every reconciliation constraint in `R` is a linear consequence of the declared constraints `O` with matching right-hand side — i.e. `R ⊆ span(O)` (`fee = 0`) and the forced value of each reconciliation is `0` (consistency). Equivalently: the declared transports and pins determine every reconciliation and force its holonomy to `0`.

Proof. `fee = 0` says `rank[M_O;M_R] = rank M_O`, i.e. each row of `M_R` lies in the row space of `M_O` — every reconciliation's coefficient vector is a combination `∑ c_i (M_O)_i`. The scalar `∑ c_i (b_O)_i` is well-defined (independent of the non-unique choice of `c_i` when the `M_O` rows are dependent): whenever `O` is satisfiable, fixing any solution `ξ` of `M_O ξ = b_O` gives `∑ c_i (b_O)_i = (∑ c_i (M_O)_i)\,ξ = M_r\,ξ`, a value determined by `M_r` alone — this is the forced holonomy of the reconciliation, well-posed exactly on the `O`-consistent branch. Consistency of `O ∪ R` then forces it to agree with the reconciliation's RHS: `M_r ξ' = 0` for a witness `ξ'` of `O ∪ R`, and since `ξ'` also satisfies `O`, `∑ c_i (b_O)_i = M_r ξ' = 0`. Conversely if either fails, `R ⊄ span O` (`fee > 0`) or some forced holonomy is `≠ 0` (clash). ■

So coherence is precisely "the declared conventions determine every reconciliation and force it to close." A clash breaks force-it-to-close (a determined holonomy is nonzero); a positive fee breaks determine (a reconciliation runs through an undeclared convention) — and only the latter is repairable by declaring more (§3.4). This is the condition Part II's progress theorem will discharge.

3.4 3.4 Coercion is disclosure; minimum coercion is the fee

A disclosure declares a line in the observable presheaf — and the line may be relational or absolute. The absolute pin `disclose_d(e, φ)` posts `x_e = φ`; the relational disclosure `disclose_d(e_1, e_2, g)` posts `x_{e_1} − x_{e_2} = g`, declaring the convention relationship between two values (e.g. "these two rates are in the same scale", `g = 0`). Both are admissible disclosures ([CD] §2.3 III) when consistent with the frames already forced below them (downward-closure), which the checker verifies; [CD] Lemma 5.3 guarantees an admissible representative at any minimal latent context. The relational form is essential: the obstruction classes are reconciliation differences `x_{e_1} − x_{e_2}`, so the canonical coercion discloses a class (a line in `H^1`), which is generally relational. Absolute pins alone do not suffice for repair duality — pinning columns can require more than `fee` disclosures when a reconciliation relates two free frames — whereas disclosing the `fee` obstruction classes always reaches coherence in exactly `fee` steps (Thm 3.7).

Proposition 3.6 (coercion = admissible disclosure). Each downward-closed disclosure (absolute or relational) induces an admissible one-generator disclosure of `G_e`; minimal-context disclosures always have an admissible representative ([CD] Lemma 5.3). ■

Elaboration repairs under-determination, not clashes. When `e` is consistent but `fee(e) > 0`, the checker elaborates `Γ ⊢ e ⇝ e' : τ` by inserting coercions until `fee = 0`. (A clash is not repaired by disclosure — declaring more frames cannot remove a contradiction; it is reported as a hard error.)

Theorem 3.7 (minimum coercion = disclosure normal form). For consistent `e`, a coercion set making `e` coherent exists; the minimum has size exactly `fee(e)`, and a minimum set is a basis of the witness matroid — the disclosure normal form; a minimum-cost set under a per-field cost is the min-weight basis, by the matroid greedy in `O(fee(e)·|e|^2)`.

Proof. `G_e ∈ SemComp_ex` (Prop 3.3) and `e` consistent, so [CD] Thm 6.2 (repair duality `ρ = r`) and Thm 7.1 (disclosure normal form) apply via Prop 3.6; each admissible nontrivial disclosure drops `fee` by one ([CD] Lemma 3.7) and (being consistent with the forced frames) introduces no clash; the matroid-greedy minimum-cost statement is [LV]'s repair theorem. ■

3.5 3.5 Frame polymorphism

`Gen_Γ` generalizes frame variables and transports not free in `Γ`. The pipe combinator is typed once:

$$\texttt{pipe} : \forall\alpha\,\forall g\,\forall g'.\,(\mathsf{b}\langle d{:}\alpha\rangle \to \mathsf{b}'\langle d{:}g{+}\alpha\rangle) \to (\mathsf{b}'\langle d{:}g{+}\alpha\rangle \to \mathsf{b}''\langle d{:}g'{+}g{+}\alpha\rangle) \to \mathsf{b}\langle d{:}\alpha\rangle \to \mathsf{b}''\langle d{:}g'{+}g{+}\alpha\rangle,$$

`pipe f g x = g (f x)` (transports `∀`-bound, determined at the call site). `pipe` posts no reconciliation, so by Prop 2.2 it generates no obstruction and instantiates soundly at any frames.

Proposition 3.8 (polymorphism hides no obstruction, in the recursion-free core). A closed, reconciliation-free combinator has `fee = 0` and is consistent at every instantiation; obstruction arises only when its result is later fed into a `reconcile`. Hence generalization is sound.

Proof. By Prop 2.2 a reconciliation-free term has acyclic `G_e` and empty `R`, so `fee = rank[M_O] − rank M_O = 0` and `O` (a triangular DAG system) is consistent. A later `reconcile` adds the constraint at its own use site. ■

(Obstruction is a property of loop-closing use sites, not of polymorphic combinators. A combinator that does close a loop — a feedback/`fix` — carries a residual coherence constraint in its scheme; that is the graded-modality content of Part III, and why the core excludes it.)

3.6 3.6 The frame-rule reading

Separation logic's frame rule is sound when the framed assertion is stable; λ_∇'s composition is sound when the conventions glue — coherent. [CD] §6.3 names the fee "the minimum disclosure required to recover frameability"; Thm 3.7 makes it operational.

3.7 3.7 The boundaries

The following are all out of scope (§1.5): (i) recursion/feedback (a `fix` closes a loop through application, breaking Prop 2.2 — Part III's graded modality); (ii) frame-dependent transports (a declared transport depending on another dimension's frame — the settlement basis, §2.5a — couples dimensions); (iii) multi-frame interfaces / non-thin coercions, where the obstruction is in H^2 (a gerbe, [SCPI]) and coercions must themselves cohere — a 2-categorical typing problem. λ_∇ is the 1-categorical, abelian, recursion-free core.

---

4 Part II. Dynamics and Soundness

5 4. Operational semantics

We give a call-by-value small-step semantics on frame-tagged values. The point of Part II is that the static coherence judgment (§3.3) genuinely guarantees the dynamic property it was designed for: a coherent program never gets stuck reconciling mismatched conventions, no matter what the opaque tools actually do.

5.1 4.1 Runtime configurations

Runtime values carry concrete frames:

$$v ::= \langle k \rangle_{d:\phi} \;\mid\; (v_1, v_2) \;\mid\; \langle \lambda x{:}\tau.e,\ \rho \rangle,$$

where `⟨k⟩_{d:φ}` is a base datum `k` tagged with a concrete frame `φ ∈ 𝔽_d` on its relevant dimension, and `ρ` is a value environment. The opaque tools' actual behavior is fixed by an opacity environment `ε`, assigning to each opaque-tool occurrence `o` and dimension `d` a concrete actual transport `ε(o,d) ∈ 𝔼_d` — what the tool really does, which the program never declared. Reduction `→_ε` is parameterized by `ε`; the soundness theorem quantifies over all `ε`.

5.2 4.2 Reduction

The standard rules (`β`, `let`, projection) are as usual. The convention-relevant rules:

- Declared tool. `t(\langle k\rangle_{d:\phi}) \to_ε \langle k'\rangle_{d:\,g^t_d + \phi}` — the output frame is the input frame transported by the declared `g^t_d`.
- Opaque tool at occurrence `o`. `t(\langle k\rangle_{d:\phi}) \to_ε \langle k'\rangle_{d:\,ε(o,d) + \phi}` — the output frame is shifted by the tool's actual transport `ε(o,d)`, which the type system did not know. (Equivalently, the opaque output's frame is an arbitrary `𝔼_d`-translate; `ε` ranges over all.)
- Disclosure. `\mathsf{disclose}_d(\langle k\rangle_{d:\phi}, \psi) \to_ε \langle k\rangle_{d:\psi}` — a trusted re-tag to the declared frame `ψ`. (Disclosure is the programmer's assertion of a convention; the type system's job is to make reconciliations succeed given the declarations. A mis-declaration is garbage-in, outside the theorem.)
- Reconciliation.

$$\mathsf{reconcile}_d(\langle k_1 \rangle_{d:\phi_1}, \langle k_2 \rangle_{d:\phi_2}) \to_\varepsilon \begin{cases} \langle k_1 \rangle_{d:\phi_1} & \text{if } \phi_1 = \phi_2, \\ \mathrm{stuck}_d & \text{if } \phi_1 \neq \phi_2. \end{cases}$$

Definition 4.1 (convention-stuck). A closed term is convention-stuck if it is not a value and its only redex is a `reconcile_d(v_1,v_2)` with `v_1,v_2` carrying unequal `d`-frames. (Every other

non-value closed well-typed term has a redex, by the standard progress argument for STLC; the convention-stuck state is the only new stuck form λ_∇ introduces.)

---

6 5. Soundness

The whole-system frame state at any point of a run is captured by a valuation: assign to each frame variable x_v (a value occurrence) the concrete frame its runtime value carries. We show this valuation always satisfies the declared constraints, and that coherence then forces the reconciliations.

Lemma 5.1 (the declared constraints hold at run time). Let e be well-typed and η the valuation induced by any reduction state of e under any ε. Then η satisfies every declared constraint in Θ (every declared-tool relation and every pin).

Proof. By cases on the reduction rules. A declared tool produces \langle\cdot\rangle_{d:g^t_d+\phi} from input \langle\cdot\rangle_{d:\phi}, so η(x_{out}) = g^t_d + η(x_{in}) — the declared relation x_{out} = g^t_d + x_{in} holds. A disclosure re-tags to ψ, so η(x_v) = ψ — the pin x_v = ψ holds. Opaque tools post no Θ-constraint, so ε's arbitrary choices constrain nothing in Θ. Substitution (β, let) carries a value with its frame intact, preserving the relations. Hence η ⊨ Θ throughout. ■

Note η need not satisfy R: an opaque tool's actual transport ε(o,d) can make a reconciliation's two frames differ. That is exactly the convention-stuck risk; coherence rules it out.

Lemma 5.2 (coherence forces the reconciliations). If ⊢ e ✓ (consistent, fee = 0) then every valuation η with η ⊨ Θ also satisfies every reconciliation in R.

Proof. By Prop 3.5, fee = 0 gives R ⊆ span(Θ): each reconciliation r ∈ R is M_r = ∑_i c_i (M_Θ)_i for scalars c_i. For any η ⊨ Θ we have (M_Θ)_i\,η = (b_Θ)_i, so M_r\,η = ∑_i c_i (b_Θ)_i. Consistency (Prop 3.5) gives ∑_i c_i (b_Θ)_i = b_r = 0. Hence M_r\,η = 0, i.e. η satisfies r. ■

Lemma 5.3 (preservation). If ⊢ e : τ ✓ and e →_ε e' then ⊢ e' : τ ✓ (for every ε).

Proof. Typing preservation is the standard STLC substitution argument, extended to the tool/disclose/ reconcile forms; we check coherence is preserved. Each step either (i) realizes a declared relation (a declared tool fires: the relation x_{out} = g + x_{in} becomes a fact about concrete frames, which only adds to Θ's consequences — R ⊆ span Θ is preserved and no clash is created since by Lemma 5.1 the realized value satisfies Θ); (ii) re-tags by a disclosure (likewise adds a pin already in Θ); (iii) discharges a reconciliation that, by Lemma 5.2, held — removing a satisfied R-constraint cannot increase fee or create inconsistency; or (iv) is a β/let/projection step, whose substitution lemma (a value of frame φ substituted for a variable of frame term α adds the pin α = φ, consistent with Θ by Lemma 5.1) preserves both R ⊆ span Θ

and consistency. An opaque tool firing adds no constraint, so trivially preserves coherence. In every case `fee(e') = 0` and `e'` consistent. ■

Theorem 5.4 (Coherence Soundness). Let $\vdash$ `e : τ ✓` be a closed coherent term. Then for every opacity environment `ε`:

1. (Progress) `e` is a value or `e →_ε e'` for some `e'`; and `e` is never convention-stuck.
2. (Preservation) if `e →_ε e'` then $\vdash$ `e' : τ ✓`.

Hence no reduction sequence of a coherent term reaches `stuck_d` — whatever the opaque tools do.

Proof. (2) is Lemma 5.3. For (1): by the standard STLC progress argument, a closed well-typed non-value term has a redex; the only redex that could fail to step is a `reconcile_d(v_1,v_2)` with `v_1,v_2` carrying unequal frames (Def 4.1). Suppose such a redex is reached. By preservation (2) the reached term is still coherent, so its reconciliation constraint `r` is in `R`. By Lemma 5.1 the runtime valuation `η` satisfies `0`; by Lemma 5.2 it then satisfies `r`, i.e. `η(x_{v_1}) = η(x_{v_2})` — the two frames are equal, contradicting the supposed mismatch. So no convention-stuck redex is ever reached, and the term steps. ■

The shape of the proof is the point. Progress is short because the statics are right: the declarations hold at run time unconditionally (Lemma 5.1), and `fee = 0` makes every reconciliation a linear consequence of the declarations (Lemma 5.2), so consistency forces it. The opaque tools' arbitrary behavior (`ε`) enters only off the forced reconciliations — exactly the content of `fee = 0`. This is the operational meaning of the companion's slogan that the fee is "the minimum disclosure required to recover frameability": with `fee = 0` the frame is recovered, and every seam closes.

Corollary 5.5 (elaboration soundness). For consistent `e` with `fee(e) = k`, the elaboration of Thm 3.7 inserts `k` coercions to yield `e'` with $\vdash$ `e' ✓`; by Thm 5.4, `e'` never gets convention-stuck. The `k` coercions are the disclosure normal form, and `k` is minimum.

Proof. Thm 3.7 produces coherent `e'`; Thm 5.4 applies. ■

6.1 5.6 What Part II establishes; what is next

λ_∇ is sound: a program the type-checker accepts (consistent, `fee = 0`) cannot, at run time, reach a convention mismatch — for any behavior of the undeclared (opaque) tools. The fee is the exact number of declarations needed to reach that guarantee. Convention coherence is thereby a genuine typing discipline, not a diagnostic: it prevents the failure rather than scoring its likelihood.

Part III admits feedback/recursion and internalizes the fee as a graded disclosure modality `□_r` (grade = fee), whose two introduction forms are [CD] §7.0's revelation and contraction, connecting the grade to graded/quantitative type theory, and proves interface-evolution-as-

type-preservation ([CD] Conj 9.5). Part IV mechanizes Thm 5.4 (reusing [CD]'s Lean carriers and the latent-complexity measure μ as the termination/measure scaffold) and extends Bulla into a prototype elaborator inserting disclosure-normal-form coercions on the MCP corpus.

Falsification. The soundness theorem fails if a coherent λ_∇ term reaches `stuck_d` under some ε (refuting Thm 5.4) — which by Lemmas 5.1–5.2 would require `R ⊄ span 0` at `fee = 0` (a rank miscomputation) or a declared constraint violated at run time (a semantics error). Either is concrete and checkable, by exhibiting the term, the ε, and the reduction trace.

---

7 Part III. The Graded Disclosure Modality and Interface Evolution

Parts I–II treat coherence as binary: `fee = 0` or not. Part III internalizes the fee as a graded modality `□_r`, with three payoffs. (1) Non-coherent terms become typeable, carrying their obligation count as a grade, so the type system measures incoherence rather than only rejecting it. (2) Recursion/feedback — excluded from the Part I core because it closes loops through application (breaking Prop 2.2) — is re-admitted, the grade carrying the feedback loop's residual obstruction. (3) The deep result: an interface update that is chain-homotopy-trivial preserves the grade and transports the receipt, discharging the abstract invariance half of [CD]'s open Conjecture 9.5 in the λ_∇ setting and yielding incremental recertification with a Lipschitz stability bound. The grade is a coeffect, but an unusual one: not a usage bound chosen by the programmer, but the witness rank forced by cohomology ([CD] Thm 5.1's uniqueness). That is what makes λ_∇ a graded type theory with a canonical grade.

8 6. The graded disclosure modality □_r

8.1 6.1 The grade monoid

Grades live in the additive ordered monoid `(ℕ, +, 0, ≤)` of disclosure obligations. Additivity is forced by the program: the fee is additive over disjoint composition ([CD] Lemma 3.5) and sums along a tower ([HF] Paper I's tower law), so composing two sub-systems adds their obligation counts.

8.2 6.2 The modality and its comonad structure

Definition 6.1. `□_r τ` is the type of a `τ` carrying `r` unresolved disclosure obligations — a value of underlying type `τ` whose convention coherence is obstructed by an `r`-dimensional witness class. Part I's coherent fragment is the grade-0 fragment: `□_0 τ ≡ τ`, and `Γ ⊢ e : τ ✓` (Part I) is exactly `Γ ⊢ e : □_0 τ`.

Proposition 6.2 (□ is a graded comonad, with a debt-ordered subsumption). Over the additive monoid `(ℕ, +, 0)`, the family `(□_r)_{r∈ℕ}` with - counit `ε : □_0 ⇒ \mathrm{Id}` (extract a

coherent value), and - comultiplication `δ_{r,s} : □_{r+s} ⇒ □_r∘□_s` (split obligations into an outer `r` and inner `s`)

is a graded comonad: `δ` is coassociative and `ε` is a two-sided counit, the laws being exactly the associativity and unit laws of `(ℕ,+,0)` (e.g. `(□_r ε)∘δ_{r,0} = \mathrm{id}_{□_r}` via `r+0=r`). This is the additive instance of the coeffect-graded comonad of Gaboardi–Katsumata–Orchard [GKO] (cf. bounded linear logic's `!_r`, [GSS]). Proof. The components are the canonical regroupings of obligation counts; each law reduces to the corresponding monoid identity in `(ℕ,+,0)`. ■

Separately, the grade is an upper bound on obligations (a debt), so there is a sound subsumption coercion `wk : □_r τ → □_s τ` for `r ≤ s` — claiming more obligations than one has is always safe, costing only a redundant disclosure (A4b). Note the direction: this is the reverse of the usage-bound monotonicity of [GKO]/[GSS] (where `□_s → □_r` for `r ≤ s`), because the λ_∇ grade is a debt to be discharged, not a capability to be spent. We therefore keep `wk` as a subtyping coercion (`□_r <: □_s`, `r ≤ s`) rather than folding it into the comonad's order structure; `reveal`/`contract` (§6.3), which lower the debt, are not part of the bare comonad since each requires a disclosure witness.

8.3 6.3 The two discharge operations (= [CD] §7.0)

A receipt is built from two structurally inequivalent operations — exactly [CD] §7.0's revelation and contraction, now typed as the moves that drive the grade to 0:

- reveal (revelation, A4a): `\mathsf{rev} : □_{r+1}\,τ → □_r\,τ`, parameterized by an admissible disclosure of a class with nonzero cohomology — discharges one independent obstruction; grade −1.
- contract (contraction, A4b): `\mathsf{ctr} : □_r\,τ → □_r\,τ`, removes a redundant latent dimension (latent complexity `μ → μ−1`); grade unchanged.

These are not interchangeable: [CD] §7.0's hybrid-semantics discovery is that revelation cannot witness A4b and contraction cannot witness A4a (verified there by `native_decide` counterexamples). A receipt is therefore a coKleisli morphism `□_r τ → □_0 τ` — a sequence of exactly `r` reveals interleaved with any number of contracts, followed by `ε`. The cardinality `r` is forced (Thm 3.7); the reveal/contract mix is the free parameter ([CD] §7.0, §6.4).

Where `contract` bites. In the exact, single-dimension core of Parts I–II, `contract` is inert: DFD+CHP force the leaf-latent group `C⁰_seam = 0` ([CD] Rmk 4.2.1), so every nontrivial disclosure has `[σ] ≠ 0` and there are no coboundary classes to contract — receipts there are pure `reveal` sequences. `contract` becomes load-bearing only once feedback (§6.6) introduces leaf-latent state (`C⁰_seam ≠ 0`), where a disclosure can re-expose inherited tool state (a coboundary) that A4b mandates leaves the grade unchanged. So the two moves are co-equal in general (the graded calculus with feedback), not in the recursion-free core.

8.4 6.4 Graded typing; the grade is the fee

The graded judgment `Γ ⊢ e : □_r τ` assigns `e` the obligation count `r`. The principal (least) grade is the fee:

Proposition 6.3. The least `r` with `Γ ⊢ e : □_r τ` derivable is `fee(e)`; and `r = 0` iff `e` is coherent (Def 3.4). Proof. `reveal` lowers the grade by exactly one per discharged independent class and `fee` counts exactly those classes (Prop 3.5); `wk` can only raise it; so the least derivable grade is `fee(e)`, zero iff coherent. ■

8.5 6.5 Connection to graded / quantitative type theory

`□_r` is a graded comonadic coeffect (Petricek–Orchard–Mycroft [POM]; Brunel–Gaboardi–Mazza–Zdancewic [BGMZ]; Orchard–Liepelt–Eades [OLE]) in the additive ordered monoid of obstruction counts, and `reveal` is its graded-dereliction step (discharge one unit). Atkey's Quantitative Type Theory [At] grades by a semiring of usages; λ_∇ grades by obstruction count. The decisive difference — and the reason this is more than an instance — is that the λ_∇ grade is canonical, not declared: in QTT/coeffect calculi the grade annotates how a programmer intends to use a variable; in λ_∇ the grade `fee(e)` is computed from the composition and is the unique invariant respecting disclosure ([CD] Thm 5.1). λ_∇ is thus a graded type theory whose grades are forced by a characterization theorem rather than chosen — a coeffect discovered in the program, not imposed on it.

8.6 6.6 Feedback and recursion, re-admitted

Part I's recursion-free core made `reconcile` the sole loop-closer (Prop 2.2). The grade re-admits feedback. A feedback former `\mathsf{loop}_d` feeds a value back through a self-map, closing a loop through application:

$$\frac{\Gamma \vdash e : \tau \to \tau \rightsquigarrow (O; R) \quad \ell := \mathit{fee} \text{ of the closed loop on } d}{\Gamma \vdash \mathsf{loop}_d(e) : \Box_\ell \, \tau} \text{ (Loop)}$$

where `ℓ ∈ {0,1}` is the loop's fee (`0` if the self-map's `d`-transport is declared and composes to the identity around the loop; `1` if opaque or unforced), generalizing to the fee of a mutual-feedback graph. So recursion is typeable, the grade tracking the feedback obstruction; Part I's `reconcile` is the special case demanding `ℓ = 0`. This restores Prop 2.2's loop-closing role to the graded setting without sacrificing soundness (§6.7).

8.7 6.7 Graded soundness: the grade is the repair cost

The grade `r = fee` does not bound the number of seams that can fail — a single undeclared convention can make arbitrarily many reconciliations stuck at once (e.g. `x₁ = opaque(x₀)` reconciled with both `x₀` and a declared copy `x₂ = x₀` gives `fee = 1` yet two simultaneously-stuck seams). What `r` bounds is the repair cost: the number of disclosures that guarantee no seam

ever fails. One `reveal` discharges one independent obstruction class and thereby un-sticks every reconciliation that depended on it. This is the operational face of repair duality ([CD] Thm 6.2): obstructions, not failing seams, are what you pay to fix.

Theorem 6.4 (Graded Soundness — repair cost). Let `Γ ⊢ e : □_r τ` be closed and consistent. Then:

1. (Progress) `e` reduces.
2. (Coherence floor) if `r = 0`, `e` is never convention-stuck under any `ε` (this is Thm 5.4).
3. (Repair) the disclosure normal form (Thm 3.7) of `r` reveals drives `e` to a grade-`0` term `e'` which, by (2), is never convention-stuck under any `ε`. Thus `r` is exactly the number of disclosures needed to guarantee no stuck seam, and a single `reveal` simultaneously certifies every reconciliation depending on the class it discharges.
4. (Tightness) `r > 0` iff some `ε` realizes a stuck reconciliation.

Proof. (1) is the progress argument of Thm 5.4. (2) is Thm 5.4. (3): by repair duality ([CD] Thm 6.2) and the disclosure normal form (Thm 3.7), `r` admissible reveals reach `fee = 0` while preserving consistency (each pin is consistent with the forced frames), and Thm 5.4 then applies to `e'`; one reveal discharges one class `[σ_i] ∈ H¹`, forcing every reconciliation whose constraint involves `[σ_i]` (Prop 3.5), of which there may be many. (4): an unforced class has a latent transport whose `ε`-instantiation can make a dependent reconciliation's holonomy nonzero (§2.5(b)); and `r = 0` is exactly (2). ■

So the grade is a cost, not a count: `r` disclosures suffice (and are necessary, by duality) to make a consistent term never-stuck, even when the number of seams those disclosures rescue is far larger. The computational QA (§5 and the Part III check) exhibits both faces: under-determined terms do get stuck (tightness, `9601/9601` in §5), and the number of stuck seams routinely exceeds `r` while `r` disclosures still repair all of them.

9 7. Interface evolution as graded type preservation (discharging the abstract half of [CD] Conjecture 9.5)

9.1 7.1 The problem

Tool manifests change: fields are added, conventions renamed, restrictions refined. Re-deriving coherence from scratch after every manifest update is the operational pain point ([CD] §9.5 calls this the most product-adjacent open problem). We want: an update that does not change the obstruction structure should preserve typing and transport the receipt — recertification for free — and a small update should change the obligation count by only a little.

9.2 7.2 Typed interface updates

Definition 7.1. A typed interface update `u : Γ ⇒ Γ'` is a morphism of convention sheaves — a natural transformation of the convention presheaves that respects the observable/latent split — inducing a chain map `u^\bullet : C^\bullet(G_e) → C^\bullet(G_{u·e})` on the seam cochain complexes ([CD] §3). The expanding updates — adding an observable field (extends `0`) and renaming a convention (an isomorphism on a stalk) — induce forward chain maps. Not every update is forward. Convention narrowing (refining a restriction so a previously-accepted convention is rejected) has no forward map in the update category; following [CD]'s Phase-B analysis of Conj 9.5 it is modeled as a span `Γ ← Γ'' ⇒ Γ'`, and receipts do not transport across it — narrowing requires a fresh (partial) audit. This is the interface-update map [CD] Conj 9.5 left to be defined; the theorem below covers the forward, chain-homotopy-equivalent case, with narrowing carved out explicitly.

9.3 7.3 The preservation theorem

Theorem 7.2 (Graded Preservation under Interface Evolution; discharges the abstract invariance half of [CD] Conj 9.5 in λ_∇). Let `u : Γ ⇒ Γ'` be a forward typed interface update (Def 7.1) whose induced chain map `u^\bullet` is a chain homotopy equivalence. Then for every well-typed `e`:

1. `fee(u·e) = fee(e)` — the grade is preserved;
2. the obstruction class transports: `u^* : H^1(G_e) ≅ H^1(G_{u·e})` is an isomorphism;
3. any receipt (disclosure normal form, Thm 3.7) for `e` transports to a receipt for `u·e`.

Hence `Γ ⊢ e : □_r τ` implies `Γ' ⊢ u·e : □_r τ` with the receipt transported — recertification needs no re-derivation.

Proof. A chain homotopy equivalence induces isomorphisms on cohomology in every degree (standard homological algebra: if `g^\bullet u^\bullet ≃ \mathrm{id}` and `u^\bullet g^\bullet ≃ \mathrm{id}` via chain homotopies, then `H^\bullet(u)` is invertible). Hence `dim H^1(G_e) = dim H^1(G_{u·e})` — (1) and (2). For (3) we argue at the level of cohomology classes, not pointwise contexts — a chain homotopy equivalence need not be a poset isomorphism and need not preserve the latent complexity μ, so it cannot be assumed to carry an admissible disclosure to an admissible one verbatim. Instead: a receipt is a basis `{[σ_i]}` of `H^1(G_e)` with chosen admissible representatives (Thm 3.7); transport the classes by the isomorphism `u^*` to a basis `{u^*[σ_i]}` of `H^1(G_{u·e})`; then re-select an admissible representative of each `u^*[σ_i]` at a minimal latent context of `G_{u·e}` (which exists by [CD] Lemma 5.3). The result is a valid receipt for `u·e` of the same cardinality `r`, so the graded type `□_r τ` is preserved. (This is [CD]'s own Phase-B route for Conj 9.5: transport the class, re-realize the disclosure.) ■

[CD] §9.5 flagged this as "a routine specialization of standard cochain-homotopy invariance; the work is in defining the right notion of interface-update map." Definition 7.1 supplies that map for the forward case; clause (3)'s class-level transport is the operational content turning invari-

ance into incremental recertification. We are deliberately modest about scope: this discharges the abstract invariance half of Conj 9.5 (the forward, chain-homotopy-equivalent updates); the narrowing half — where [CD]'s Phase B shows there is no forward map and receipts genuinely do not transport — is carved out in Def 7.1 and remains, with [CD], a non-transporting span requiring partial re-audit.

9.4 7.4 Stability: bounded recertification under general updates

Most real updates are not chain-homotopy equivalences — they genuinely add or remove obstructions. For these we want a stability bound, so recertification is still local.

Definition 7.3. An update is `k`-local if it modifies the convention sheaf only on a sub-cover touching at most `k` convention generators (it adds or deletes at most `k` cochain dimensions).

Theorem 7.4 (Stability / bounded recertification). For a `k`-local update `u`,

$$|\, fee(u \cdot e) - fee(e) \,| \leq k.$$

Consequently a coherent `e` remains coherent after at most `k` new disclosures, and the recertification cost is `O(k · |\text{update}|)`, independent of `|e|`.

Proof. `fee = dim H^1` is the rank of a quotient of cochain groups; adding or deleting `k` generators edits the coboundary by at most `k` rows/columns, and matrix rank is `1`-Lipschitz under a rank-1 row/ column edit, so `dim H^1` changes by at most `k`. Equivalently, by excision / Mayer–Vietoris (axiom A6, [CD]), `fee(u·e) − fee(e)` is determined by the change on the touched sub-cover and its overlap, bounded by its generator count `k`. ■

This is the persistence-stability analogue [CD] §9.5 points to (Cohen–Steiner–Edelsbrunner–Harer [CSEH]): the fee is `1`-Lipschitz in the interface under generator-edit distance, just as a persistence barcode is stable in bottleneck distance under perturbation of the underlying data. It licenses a coherence-preserving versioning discipline: ship each manifest update with its `k`-local chain data; recertify by adjusting at most `k` disclosures, never re-auditing the whole composition. One caveat, inherited from §7.3: a convention-narrowing update (a tool tightening which conventions it accepts) is breaking — receipts do not transport across it — so the discipline carves narrowing out as the case that triggers a fresh, but still `k`-local (Thm 7.4), partial audit rather than a free transport. For the agent-systems setting this is the decisive practical property — tool manifests churn continuously, and Thm 7.4 bounds the audit cost of churn (forward or narrowing) by the size of the change, not the size of the system.

9.5 7.5 What Part III establishes

The fee is realized as a graded comonadic modality `□_r` (grade = witness rank), with revelation and contraction as its two discharge operations ([CD] §7.0) and a graded soundness theorem (Thm 6.4: the grade is the repair cost — `r` disclosures guarantee no stuck seam, and `r = 0` ⇒

never stuck — not a bound on the number of seams that can fail, since one undeclared convention can break many seams at once and one `reveal` repairs them all). Recursion/feedback is readmitted, the grade carrying the loop obstruction. Interface evolution that is chain-homotopy-trivial preserves the grade and transports the receipt — discharging the abstract invariance half of [CD] Conjecture 9.5 in the λ_∇ setting (with convention-narrowing handled as a non-transporting span, §7.3) — and general k-local updates change the grade by at most k (stability), giving incremental recertification bounded by update size. Part IV mechanizes Thms 5.4 and 7.2 (reusing [CD]'s Lean carriers, the μ-induction, and the chain-homotopy machinery) and extends Bulla into a prototype elaborator that inserts disclosure-normal-form coercions and performs k-local recertification on the MCP corpus.

Falsification. Part III fails if a chain-homotopy-equivalent update changes the fee (refuting Thm 7.2 — which by cohomology-invariance would expose a definitional error in Def 7.1's interface-update map), or if a k-local update changes the fee by more than k (refuting Thm 7.4). Both are concrete: exhibit the update, its chain data, and the before/after fees.

---

10 References

[CD] J. Komkov, A Witness Logic for Semantic Composition, Res Agentica program, 2026.

[LV] J. Komkov, Local Validity Does Not Compose, Res Agentica program, 2026.

[PAR] J. Komkov, Compositional Incoherence is a Parity, Res Agentica program, 2026.

[HF] J. Komkov, The Coherence Fee and Hierarchical Decomposition of Coherence Fee, Res Agentica program, 2026. (DFD+CHP sufficiency for the exact regime.)

[SI] J. Komkov, Signed-Incidence Structure in Compositional Verification, Res Agentica program, 2026.

[SCPI] J. Komkov, SCPI: Predicate Invention Under Sheaf Constraints, Res Agentica program, 2026.

[1] C. A. R. Hoare, "An axiomatic basis for computer programming," CACM 12(10) (1969), 576–580.

[2] J. C. Reynolds, "Separation logic," LICS 2002, 55–74.

[3] R. Milner, "A theory of type polymorphism in programming," JCSS 17(3) (1978), 348–375.

[4] A. K. Wright and M. Felleisen, "A syntactic approach to type soundness," Information and Computation 115(1) (1994), 38–94.

[5] J. Hansen and R. Ghrist, "Toward a spectral theory of cellular sheaves," J. Appl. Comput. Topol. 3(4) (2019), 315–358.

[At] R. Atkey, "Syntax and semantics of quantitative type theory," LICS 2018, 56–65.

[GKO] M. Gaboardi, S. Katsumata, D. Orchard, F. Breuvart, T. Uustalu, "Combining effects and coeffects via grading," ICFP 2016, 476–489.

[POM] T. Petricek, D. Orchard, A. Mycroft, "Coeffects: a calculus of context-dependent computation," ICFP 2014, 123–135.

[BGMZ] A. Brunel, M. Gaboardi, D. Mazza, S. Zdancewic, "A core quantitative coeffect calculus," ESOP 2014, LNCS 8410, 351–370.

[OLE] D. Orchard, V.-B. Liepelt, H. Eades III, "Quantitative program reasoning with graded modal types," Proc. ACM Program. Lang. 3 (ICFP) (2019), Article 110.

[GSS] J.-Y. Girard, A. Scedrov, P. J. Scott, "Bounded linear logic: a modular approach to polynomial-time computability," Theoretical Computer Science 97(1) (1992), 1–66.

[CSEH] D. Cohen-Steiner, H. Edelsbrunner, J. Harer, "Stability of persistence diagrams," Discrete & Computational Geometry 37(1) (2007), 103–120.