

Normalization confluence: an exact, mechanized theory of when repair can replace commutativity

Dayna Blackwell

October 2026

Abstract

Replicas that apply the same events in different orders, and repair their business invariants after each event, need not end in the same state. We give an exact theory of when they do: raw states may differ, but the result after repair must agree. We state a condition at each of three scales: state descent, which makes the result independent of when repair runs; history descent over finite histories, which with state descent and terminating repair is necessary and sufficient for a single registry; and canonical recurrence, necessary and sufficient for infinite runs on finite state spaces. We prove the hierarchy between them and show that CRDTs are the case where repair does nothing: repair enlarges which operations can be used, while every convergent repaired system is a CRDT on its repaired steps. A three-layer framework for executions, consistent states and long-run behavior, with proved interfaces, organizes these conditions across a map of regimes that has, in every regime it does not exclude by design, an exact condition, a hardness result or a listed open problem. We test the framework where it could fail: with its files frozen it recovers Dijkstra’s self-stabilization and invariant confluence, and two predictions committed before their tests were confirmed, one about stable matching markets and one about replication in CouchDB and PouchDB. All 4,797 gated results are mechanized in Coq with no axioms.

1 Introduction

Replicated systems deliver the same events to different replicas in different orders, because networks delay, reorder and retry messages. Real applications also have rules: an order ships only once it is paid, a balance stays above a floor, a shelf holds at most so many items. In practice, systems keep such rules by repairing after the fact. An event is applied, and if it breaks a rule, a deterministic repair restores it: a saga compensates, a constraint is re-established, a value is capped. The question this paper answers is when that works: when replicas that have seen the same events and run the same repair end in the same state, whatever order the events arrive in and whenever the repair runs.

It does not always work. A shelf holds at most 5 items and has 4 in stock; the events “add 3” and “remove 1” arrive, and the repair caps the stock at 5. One replica applies “add 3”, caps at 5 and removes one, ending at 4; another removes one, adds three and caps, ending at 5. The same events and the same repair give different results.

The two established answers to concurrency sidestep the question. CRDTs [1] design operations so that order never matters, and need no repair. Invariant confluence [2] asks which operations preserve an invariant without coordination, with the invariant checked and never repaired. For compensation as practiced, in sagas [3] and in systems that add invariants to weakly consistent replication [4, 5], we found no exact condition in that literature for when concurrent orders converge.

The answer in one sentence. Raw states may differ along the way; what has to agree is the result after repair. The paper makes this precise at three scales: for a single event, for finite histories, and for infinite runs.

Contributions.

1. Conditions at three scales, with a proved hierarchy between them: state descent, necessary at one event when events can arrive before repair finishes; history descent, which with state descent and terminating repair is exact for a single registry; and canonical recurrence, exact for infinite runs on finite state spaces (Section 3).
2. The relation to CRDTs stated precisely: repair strictly enlarges which operations can be used, but every convergent repaired system is a CRDT on its repaired steps; and the relation to invariant confluence (Section 4).
3. A three-layer framework, for executions, consistent states and long-run behavior, with proved interfaces and the counterexamples that mark where each stops, from which the single-registry theorem is derived without circularity (Sections 5 to 9).
4. Tests of the framework where it could fail: classical theorems from outside the development recovered with its files frozen, and two predictions committed before their tests, one a bet about matching markets and one a computed prediction about widely used databases, both confirmed (Section 10).
5. A regime map with an exact condition, a hardness result or a listed open problem in every regime not excluded by design, all mechanized in Coq with no axioms, and a checker that runs the practical conditions at build time (Sections 11 and 12).

Scope. “Exact” means necessary and sufficient relative to the model each result states; counterexamples fix the qualifiers they name. The mathematical tools, Newman’s lemma, trace theory, constraint-satisfaction algebra and fair cycles, are classical and credited where they are used (Sections 13 and 14).

2 Setting

Registries. A *registry* fixes a set of states X , a validity predicate **valid** (the business rules), a set of events, and a repair. Each event e has a raw transition $e : X \rightarrow X$, which may leave the valid states. The repair is given by a step $\rho : X \rightarrow X$ applied to invalid states, with a potential $\Phi : X \rightarrow \mathbb{N}$ that strictly decreases under ρ on invalid states, so that repair terminates. This termination condition is called WFC (well-founded compensation). Iterating ρ until the state is valid gives the repair map $N = \rho^*$, and the governed (repaired) step of an event is $\text{gov}(e) = \rho^* \circ e$.

Executions. A replica holds a configuration: a state and a buffer of pending events. A step either applies a pending event or, if the state is invalid, applies ρ . A normal form is a valid state with an empty buffer. A state is reachable from a start s_0 when some sequence of event and repair steps leads to it; the exact conditions of the paper quantify over reachable states.

Delivery models. Which orders a replica may see is fixed by a delivery model: free delivery (any order), causal delivery (orders consistent with a happens-before relation), at-least-once delivery (also duplicates), and declared independence (only declared pairs may be swapped). Section 5 treats them uniformly.

Networks of registries. A federation connects registries by transport maps along which values propagate; a network may be acyclic or contain cycles, and propagation may be lossy. These are the settings of layer B and of most of the regime map; their definitions are introduced where they are used (Sections 7 and 11).

Mechanization. Every result named in the paper is a theorem in a Coq development, cited by its name in typewriter font. Every theorem is closed under the global context, with no axioms and no admitted proofs, and continuous integration checks this on three versions of Coq and Rocq on every change. The statements in the paper are read from the source. Decidability assumptions (decidable equality of states and events, decidable validity) are stated where the Coq statement needs them for a constructive proof; other hypotheses are stated in full. Appendix A describes the method and Appendix B the files.

Notation	Meaning
X , valid	states and the validity predicate
$e : X \rightarrow X$	the raw transition of an event
ρ, Φ	a repair step and its potential (WFC: Φ decreases on invalid states)
$N = \rho^*$	the repair map: apply ρ until valid
$\text{gov}(e) = N \circ e$	the governed step
$\text{reach}(s_0, \sigma)$	σ is reachable from s_0
CC1, CC2	the two commutation conditions of Theorem 3.6

3 The main result

Replicas apply the same events in different orders, because networks delay, reorder and retry messages, and every replica runs the same repair after each event to restore its invariants. The question is when replicas that have seen the same events end in the same state. The answer has one shape at every scale: raw states may differ along the way, and what has to agree is the result after repair. This section states a condition at each of three scales, a single event (Section 3.1), finite histories (Section 3.2) and infinite runs (Section 3.4), together with the single-registry theorem that combines the first two (Section 3.3). When events can arrive before repair finishes, state descent is necessary; together with history descent and terminating repair it is exact for a single registry; and canonical recurrence is exact for infinite runs on finite state spaces. The three conditions are not one equation. They form a proved hierarchy, with counterexamples showing where each level stops (Section 3.5).

Throughout, X is a set of states and $N : X \rightarrow X$ is the repair. An event is a map $e : X \rightarrow X$ on raw states. A replica applies events and repairs invalid states. In the single-registry theorem an event may arrive while repair is still in progress, so a step is either an event or one repair step (Section 2). When repair always finishes before the next event, the step is the governed step $x \mapsto N(e(x))$.

3.1 One event: state descent

Definition 3.1 (State descent; **DescendsOn**). An event e *descends* on a set $P \subseteq X$ when

$$N(e(N(x))) = N(e(x)) \quad \text{for every } x \in P.$$

The condition says that applying the event to a state or to its repaired form gives the same result after repair. Repairing early loses nothing the event can observe. For an idempotent repair the

definition is the standard compatibility of a map with the kernel of N (`descends_iff_respects`); what matters here is that convergence forces it, at the states a system can reach.

Example 3.2 (A shelf). A shelf holds at most 5 items and has 4 in stock. The events are “add 3” and “remove 1”, and the repair caps the stock at 5, so $N(x) = \min(x, 5)$. Replica A applies “add 3” (7), repairs (5), then “remove 1”: it ends at 4. Replica B applies “remove 1” (3), then “add 3” (6), and repairs: it ends at 5. The two replicas saw the same events and ran the same repair, and they disagree.

State descent locates the failure. Take the raw state 7, just after “add 3”, and let e be “remove 1”. Then $N(e(N(7))) = N(e(5)) = N(4) = 4$, while $N(e(7)) = N(6) = 5$. The event does not descend at 7: removing an item before or after the cap is applied gives different answers.

In the single-registry setting of Section 3.3, state descent is exactly one of the two conditions convergence requires. There, the repair is built from a repair step ρ that strictly decreases a potential $\Phi : X \rightarrow \mathbb{N}$ on invalid states, and $N = \rho^*$ applies ρ until the state is valid.

Proposition 3.3 (`state_descent_iff_cc2`). *If validity is decidable and $\Phi(\rho(s)) < \Phi(s)$ for every invalid s , then every event descends on all of X under $N = \rho^*$ if and only if*

$$\rho^*(e(s)) = \rho^*(e(\rho(s))) \quad \text{for every event } e \text{ and every invalid } s.$$

The right-hand side is the condition called CC2 below: an event does not care whether one repair step ran before it.

3.2 Finite histories: history descent

A delivery model says which reorderings of a history count as the same delivery. Under free delivery every permutation is allowed; under causal delivery, only orders consistent with a happens-before relation; under at-least-once delivery, also duplicates; under declared independence, swaps of declared pairs.

Definition 3.4 (History descent). For a delivery model, given as an equivalence $\equiv$ on histories, and a start s_0 , the system satisfies *history descent* when every two equivalent histories give the same result after repair: $N(\text{run}(h, s_0)) = N(\text{run}(k, s_0))$ whenever $h \equiv k$, where run applies the raw events in order.

The raw runs in this definition never repair. A replica, however, repairs after every event. The next lemma says that, given state descent, the two views agree, so a checker may work with repaired states only.

Lemma 3.5 (`history_from_descent`). *Suppose every event descends on X . Write $\text{grun}(h, x)$ for the run that applies the events of h to x and repairs after each one. Then for every equivalence $\equiv$ on histories and every start s_0 ,*

$$(\forall h \equiv k. N(\text{run}(h, s_0)) = N(\text{run}(k, s_0))) \iff (\forall h \equiv k. \text{grun}(h, N(s_0)) = \text{grun}(k, N(s_0))).$$

The proof is the observation that, under state descent, $N(\text{run}(w, s_0)) = \text{grun}(w, N(s_0))$ for every word w (`normalization_descent`). Which reorderings preserve the result depends on the delivery model; Section 5 gives one theorem pair that decides it for each model above. Here only free delivery is needed, through the single-registry theorem.

3.3 The single-registry theorem

We use the registries, configurations and reachability of Section 2, under free delivery: every pending event may fire.

Theorem 3.6 (Compensation commutativity; `cc_exact`). *Assume free delivery and decidable equality of events; that $\Phi(\rho(s)) < \Phi(s)$ for every invalid s (WFC); that $\rho^*(\sigma)$ is valid for every σ ; and that $\rho^*(\sigma)$ is reached from σ by repair steps. For every start s_0 , every configuration (σ, B) with $\text{reach}(s_0, \sigma)$ has a unique normal form if and only if both of the following hold at every state σ reachable from s_0 :*

(CC1) $\text{gov}(e_2)(\text{gov}(e_1)(\sigma)) = \text{gov}(e_1)(\text{gov}(e_2)(\sigma))$ for all events e_1, e_2 ;

(CC2) $\text{gov}(e)(\sigma) = \text{gov}(e)(\rho(\sigma))$ for every event e , when σ is invalid.

In words: replicas converge exactly when repaired events commute (CC1) and an event does not care whether repair ran first (CC2), at the states they can reach. Termination of repair is the standing hypothesis; it is the third condition of the theory (WFC, Section 2), as in the earlier papers [6, 7].

The theorem reads in the vocabulary of the two descents. CC2 is state descent on the reachable states, and, given state descent, CC1 is history descent for free delivery, stated on the governed steps (Section 9).

Corollary 3.7 (`nd_derived_from`). *Under the hypotheses of Theorem 3.6, every configuration (s_0, B) has a unique normal form if and only if the governed runs from $N(s_0)$ agree on every permutation of every history, and every event descends on the states reachable from s_0 .*

Section 9 shows that this form is derived from the framework’s kernels without using Theorem 3.6.

When repair runs. Restated with `atomic_derived_from`, Corollary 3.7 says that a registry converges under the model of Section 2 exactly when it converges, from the repaired start, with repair run to completion after every event, and every event descends at the reachable states. State descent is therefore what makes the result independent of when repair runs. Under atomic repair it is not needed: `atomic_not_descent` gives a registry that converges under atomic repair but has an event that does not descend at a reachable invalid state. Under atomic repair, convergence is exactly commutation of the governed steps at the states reachable by governed steps (`atomic_exact`).

Proof idea, and what is not new. Sufficiency is Newman’s lemma restricted to the reachable states, with a case split on the two kinds of critical peak: two events, and an event against a repair step. Necessity uses two small runs. The buffer $[e_1, e_2]$ in both orders forces CC1; the buffer $[e]$ with the event first versus the repair first forces CC2. Both ends of each run are normal forms, because ρ^* always reaches a valid state. The result is therefore the critical-pair theorem of Knuth and Bendix [8] and of Huet [9], specialized to a system whose critical pairs are already normal forms: that is why “joinable” becomes “equal” and the converse costs nothing. The contribution is the condition and its necessity at the reachable states, not the proof technique. Two counterexamples fix hypotheses of the theorem: the repair must reach a valid state, or the converse fails (`rho_star_qualifier`); and delivery must be free, since under causal enabledness a failure of CC1 can be masked (`masked_cc1`).

3.4 Infinite runs: canonical recurrence

When repair may never stop, for example on networks of registries that feed each other around a cycle, the question becomes one of long-run behavior. A system now has a finite set of actions J , each with an update $u_j : X \rightarrow X$, and a schedule $\tau : \mathbb{N} \rightarrow J$ picks the action at each step. A schedule is *fair* when every action is picked infinitely often, an action that changes nothing at the current state being a no-op. At this scale N may be any observation $N : X \rightarrow C$ of the state, for example its repaired result. A run *settles* on $c \in C$ when $N(x_n) = c$ for all large enough n .

A *bad fair lasso* from s_0 for c is a path from s_0 to a state x followed by a nonempty loop from x back to x that picks every action, with some state on the loop outside the fiber $N^{-1}(c)$.

Theorem 3.8 (Canonical recurrence; **C1_closed**). *Let X be finite, with decidable equality on X and on C , and suppose the fiber $N^{-1}(c)$ is closed under every update. Then every fair run from s_0 settles on c if and only if there is no bad fair lasso from s_0 for c .*

Settling fails exactly through a fair loop that leaves the target result. The kernel is the classical fair-cycle characterization of Emerson and Lei [10], read through N ; its role here is to classify how runs fail (oscillation, livelock, settling on the wrong state) in one vocabulary (Section 6). Finiteness is needed for the exact form only: a bad fair lasso refutes settling on any state space (**C1_refutes**).

At this scale the condition reads repair only through its result.

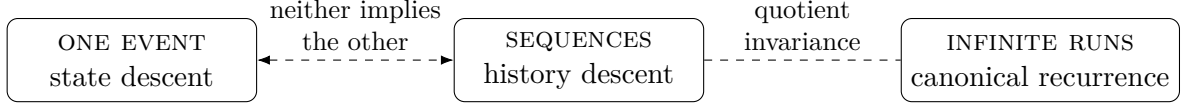
Proposition 3.9 (**canon_settles_quotient_invariant**). *Suppose the updates u_j respect N , that is $N(x) = N(y)$ implies $N(u_j(x)) = N(u_j(y))$, and let u'_j satisfy $N(u'_j(x)) = N(u_j(x))$ for every action and state. If $N(s'_0) = N(s_0)$, then every fair run of u from s_0 settles on c if and only if every fair run of u' from s'_0 does.*

3.5 How the conditions relate

The three conditions are not one equation. State descent and history descent are independent of each other; together, with terminating repair, they are exactly convergence of a single registry (Corollary 3.7). The infinite-run condition reads repair only through its result (Proposition 3.9), and without descent it is not determined by the repaired updates alone (Proposition 3.10(c)); a combined statement across the three scales is open. Each separation below is witnessed by a mechanized counterexample, and Fig. 1 summarizes the relations.

Proposition 3.10.

- (a) State descent without history descent (**descent_not_history**). *With N the identity every event descends, and two events that do not commute give different results for the two orders of one history.*
- (b) History descent without state descent (**history_not_descent**). *With one event, the successor map on $\mathbb{N}$, and the repair N that sends 0 and 1 to 0 and fixes every other number, every re-ordering of a history is the history itself, so history descent holds; but $N(e(N(1))) = N(1) = 0$ while $N(e(1)) = N(2) = 2$.*
- (c) Infinite runs need descent (**trajectory_needs_descent**). *There are two systems whose updates have the same image under N at every state, neither of which respects N , such that, from the same start, every fair run of one settles on *true* and every fair run of the other settles on *false*.*



together, with terminating repair: exactly the single-registry theorem

Figure 1: The three conditions and the proved relations between them (Propositions 3.9 and 3.10 and Corollary 3.7).

When events can arrive before repair finishes, state descent is necessary; together with history descent and terminating repair it is exact (Theorem 3.6 and Corollary 3.7); canonical recurrence is exact on finite state spaces (Theorem 3.8). The counterexamples above fix the qualifiers they name. Section 4 reads the single-registry theorem against CRDTs, the case where repair never runs, and Sections 5 to 8 give the three-layer framework that organizes these conditions across the regimes of the map.

4 CRDTs, invariant confluence, and what repair adds

Two bodies of work answer neighboring questions. Conflict-free replicated data types (CRDTs) obtain convergence by designing operations so that order never matters [1]. Invariant confluence asks which transactions can run without coordination while preserving an invariant [2]. This section places normalization confluence against both, precisely: CRDTs are the case where repair does nothing (Section 4.1); repair enlarges which operations can be used, but not which behaviors converge (Section 4.2); and invariant confluence is a different question, because there the invariant is checked and never repaired (Section 4.3).

4.1 CRDTs are the case where repair does nothing

In the terms of Section 3, a CRDT is a system whose repair N is the identity. State descent, $N(e(N(x))) = N(e(x))$, then holds for every event, so convergence comes down to history descent alone: every reordering the delivery model allows must give the same state. The classical CRDT conditions are what this reduces to under the two delivery models CRDTs are built for. As in Section 2, $\text{gov}(e) = N \circ e$ is the governed step.

Theorem 4.1 (The compensation-free boundary; `crdt_boundary` (a), (b), (b')). *Let N be the identity.*

- (a) Causal exactly-once delivery. *For every irreflexive happens-before relation, the governed steps converge causally from every start (any two causally consistent orders of the same events reach the same state) if and only if concurrent raw operations commute at every state.*
- (b) Unordered at-least-once delivery. *For every event range R with decidable equality and every start s_0 , any two deliveries over R with the same set of events, in any order and with any duplication, reach the same state from s_0 if and only if the raw operations are commutative and idempotent at every state reachable from s_0 .*
- (b') The semilattice form. *If moreover R is finite and state equality is decidable, the convergence of (b) holds if and only if the raw operations have a join-semilattice representation on the states reachable from s_0 : some join is a semilattice there and every event acts as $e(s) = s \sqcup e(s_0)$.*

Part (a) is the op-based (CmRDT) condition of Shapiro et al. [1] in exact form, stated globally: from every start, commutation at every state. Parts (b) and (b') are exact on the states reachable from s_0 , and (b') is the merge-action form of the state-based (CvRDT) condition, in which an event acts by joining a fixed payload. Shapiro et al. give their conditions as sufficient; the theorem shows these forms are also necessary. Shapiro et al.'s model of replicas that merge whole states is treated in Section 10.1, where that merge step leaked. The qualifiers are needed: reachability cannot be dropped (`clamp_reach_qualifier`), a semilattice on all of the state space is too strong (`naive_cvrtdt_iff_fails`), and both commutativity and idempotence are needed in (b) (`lww_not_conv`, `add_cc_not_alo`). The exact conditions behind (a) and (b) are `causal_convergence_exact` and `merge_action_exact`; the theorem instantiates them at the identity repair.

Proof idea. With N the identity the governed step is the raw step (`gstep_id`), so each part is an exact condition for an arbitrary step function, read at the raw operations. For (a), `causal_convergence_exact`: necessity runs two concurrent events in both orders from an arbitrary start, and both orders are causally consistent because happens-before is irreflexive; sufficiency is `causal_convergence`, which moves each event of one order to the front of the other past events concurrent with it. For (b), `merge_action_exact`: necessity compares $u \cdot [x, y]$ with $u \cdot [y, x]$, and $u \cdot [x, x]$ with $u \cdot [x]$, after a delivery u that reaches the state; sufficiency replaces each delivery by its duplicate-free form, which needs decidable equality of events (`mergeconv_alo`), and applies the exact at-least-once theorem `alo_exact`. For (b'), `cvrtdt_exact_all`: a semilattice representation gives commutativity and idempotence by the semilattice laws (`cvrtdt_on_conv`); conversely, the join of reachable states s and t runs from s a sublist of the finite range whose run from s_0 reaches t . Such a sublist exists by convergence (the events of a delivery reaching t , listed once), is found by search using decidable state equality, and the semilattice laws then follow from convergence.

4.2 Two readings of “repair enlarges what converges”

The usual claim for a repair-based design is that it reaches systems CRDTs cannot. The claim is true in one reading and false in another, and the paper separates them.

Operations: strict. Some raw operations converge only with repair.

Proposition 4.2 (`crdt_boundary` (c)). *Take a single Boolean whose only valid value is false, two events with raw transitions $\text{settrue}(s) = \text{true}$ and $\text{flip}(s) = \neg s$, and the repair that resets to false. The governed system converges under causal exactly-once delivery from every start and under unordered at-least-once delivery from every start. The raw operations satisfy none of the conditions of Theorem 4.1: they do not commute where the two events are concurrent, the system built from them without repair fails to converge under both delivery models, they are neither commutative nor idempotent at a reachable state from any start, and they have no semilattice representation on the reachable states.*

So no system without repair that uses these raw transitions converges, while the repaired system does. The separation is sharpened in `witness_ops_not_commute`, `witness_raw_not_causal`, `witness_not_cvrtdt_order` (no partial order on the Booleans makes both raw operations inflationary) and `witness_not_cvrtdt_exact`.

Proof idea. Everything is a computation on the Booleans. The governed step is the constant false (`wgov_false`), so governed steps commute and are idempotent at every state, and the sufficiency

directions of `causal_convergence_exact` and `merge_action_exact` give convergence under both delivery models (`witness_governed_converges`). On the raw side, `flip(settrue(s)) = false` and `settrue(flip(s)) = true` for every s , which breaks commutation for the concurrent pair; part (a) of Theorem 4.1 at the identity repair turns this into divergence of the system without repair. From any start s_0 , which is reachable by the empty delivery, the same pair fails to commute and `flip` fails to be idempotent at s_0 itself; `merge_action_exact` turns this into divergence under at-least-once delivery, and a semilattice representation is ruled out because it would imply convergence (`cvrtdt_on_conv`).

Behavior: not strict. The exact conditions behind Theorem 4.1 hold for any step function, not only for raw operations. Read at the governed step $\text{gov}(e) = N \circ e$, they say:

Corollary 4.3 (`causal_convergence_exact`, `merge_action_exact`, at the governed step). *For any repair N :*

- (a) *for an irreflexive happens-before relation, the governed system converges causally from every start if and only if governed steps of concurrent events commute at every state;*
- (b) *for an event range with decidable equality and a start s_0 , the governed system converges under unordered at-least-once delivery if and only if the governed steps are commutative and idempotent at every state reachable from s_0 .*

Proof idea. There is no separate proof: `causal_convergence_exact` and `merge_action_exact` are stated for an arbitrary step function, and the corollary instantiates them at $\text{gov}(e) = N \circ e$. No property of N is used, which is why the corollary holds for every repair.

These are the CmRDT and the commutative-idempotent conditions, stated for the governed steps. So every convergent governed system is a CRDT on its own governed steps, at the states it reaches. Here a CRDT means the op-based condition under causal delivery, and under at-least-once delivery the commutative and idempotent condition, which over a finite event range is the semilattice form of Theorem 4.1(b') read at the governed step (`cvrtdt_exact_all`). It does not mean a state-based CRDT whose replicas merge whole states, a model outside the framework (Section 10.1). The witness of Proposition 4.2 is an instance: its governed behavior is the constant false, and its governed steps are a CvRDT with `join` $\wedge$ (`witness_governed_constant`).

What repair adds. Normalization confluence does not reach convergent behaviors that CRDTs cannot. What it adds is a way to build them. A designer writes the operations the application needs, which need not commute, and a repair that restores the invariants; the exact conditions of Section 3, state descent and history descent, decide whether the combination converges; when repair runs with the event, that is whether its governed steps form a CRDT. The design freedom is in the operations, and the guarantee is checked rather than designed in. Section 12 describes a checker for that case that runs at build time.

The CRDT reading treats an event and its repair as one atomic operation, so it says nothing about when repair runs. When an event can arrive before repair finishes, state descent is the further condition (Section 3.3, “When repair runs”); `atomic_not_descent` gives a system whose governed steps converge under atomic repair but which fails it. The extension is in the operations and the execution model a designer may use, not in the class of order-independent outcomes.

4.3 Invariant confluence: checked, not repaired

Invariant confluence [2] asks whether a set of transactions can run on replicas that merge their states without coordination and still preserve an invariant I . A transaction commits only if its local result satisfies I , and a merge can never be refused. The invariant is checked, never repaired. This is a different question from the one normalization confluence answers, and the two meet in one place: the merge.

Model the invariant as the solution set of a constraint problem and the merge as an operation applied item by item. In the constraint-satisfaction terms of Section 7, the merge is then a binary operation on states, and invariant confluence becomes a statement about it.

Theorem 4.4 (`pol_iconfluent`, `iconf_iff_pol_reach`).

- (a) *If I holds at the start and the item-wise merge is a polymorphism of the invariant’s constraint language, every set of transactions is invariant confluent.*
- (b) *Exactly: when I holds at the start and reads finitely many items, a set of transactions is invariant confluent if and only if the merge is a polymorphism of the relation of states reachable under those transactions with I preserved.*

Proof idea. For (a), the standard fact that a polymorphism of a template maps solutions of any instance over it to a solution (`pol_solutions`) says that the merge of two I -valid states is I -valid (`pol_iclosed`). Every state reachable under the transactions is I -valid when I holds at the start (`itr_valid`), so this closure gives invariant confluence for any set of transactions (`iclosed_iconf`). For (b), when I holds at the start, invariant confluence is equivalent to closure of the reachable states under the merge (`iconf_iff_reach_closed`): a merge of reachable states with a valid result is itself reachable, and reachable states are valid. The polymorphism condition on the relation of a state predicate restricted to n items says that for any two states satisfying the predicate, some state satisfying it agrees with their merge on those items (`preserves_prel`). The hypothesis that I reads only those items is used in the converse direction, to pass validity from that state to the merge itself.

The restriction to reachable states is necessary, as Whittaker and Hellerstein [11] observed. A mechanized counterexample makes it concrete (`uq_not_pol`): with two versions a and b , merge by set union, the invariant “not both present” and the single transaction “add a ”, the transactions are invariant confluent, but union does not preserve the invariant on all states. Bailis et al.’s main theorem says that a coordination-free, available, convergent and invariant-preserving execution exists exactly for invariant-confluent transactions. What is mechanized is narrower: in a replica model, the protocol in which a server commits a transaction only if its local result satisfies I and merges every state it receives keeps every server I -valid in every execution if and only if the transactions are invariant confluent (`replica_exact`). The general theorem, over arbitrary protocols, would need a model of partitions that the development does not have. *Proof idea.* Sufficiency is an induction over the execution showing that every server always holds a state reachable under the transactions (`rstep_itr`): a committed transaction has a valid result, and a merge of two reachable states is valid by invariant confluence. Necessity replays any two reachable states on disjoint sets of servers (`replay`, Bailis et al.’s Lemma 1) and then merges them on one server (`replay_merge`), so a merge that breaks I shows up in some execution.

Repair changes the picture. With the same invariant and both transactions “add a ” and “add b ”, the transactions are not invariant confluent. With the repair $N(x, y) = (x, y \wedge \neg x)$, which drops b whenever a is present, state descent and history descent both hold, every repaired state satisfies

the invariant, and replicas that exchange repaired states agree with the repaired run of the joint history (`repair_enlarges`). The caveat is that this repair can undo a committed write, which Bailis et al.'s model forbids; the comparison is between two models, not a strengthening of theirs.

5 Layer A: execution

Layer A asks which executions count as the same. Every delivery model in the regime map answers that question differently: free delivery allows every reordering, causal delivery only those consistent with happens-before, at-least-once delivery also identifies a duplicate with its first landing, and a live reconfiguration identifies an event before the switch with its translation after it. This section states the question once, in a form every model fits, and proves the two principles that answer it. The exact history results of the map are then instances, recovered with their original statements.

5.1 Presented executions

Executions are words over a set of labels L , but not every word is an execution: causal consistency, typed phases and legal duplicate landings rule some out. The data of a delivery model is therefore a *presented admissible path system*:

- a predicate `Adm` on words, the admissible executions, closed under prefixes where a theorem needs it (`AdmPrefix`);
- generator pairs $\text{Gen}(p, u, v)$: at the point p , the fragments u and v are interchangeable, for example a swap of independent events $[a; b] \sim [b; a]$, a legal duplicate landing $[a; a] \sim [a]$, or a switch square between typed phases;
- the generated equivalence `PEq`: the equivalence closure, through admissible words only, of the whiskered pairs $p \cdot u \cdot w \sim p \cdot v \cdot w$.

Free delivery is the degenerate case: with every word admissible and generators independent of the point, `PEq` is the monoid congruence the pairs generate (`free_congruence`), and for adjacent swaps it is equality up to permutation (`swap_peq_perm`). This is the quotient a string presentation defines, as in trace theory [12, 13].

A semantics `sem` on words is compared by an equivalence $\approx$; for a state machine, `sem(t)` is the repaired state after running t from the start. *Invariance* (`PInv`) says that `sem` is constant up to $\approx$ on each `PEq` class. This is history descent (Definition 3.4) for the delivery model the presentation describes.

5.2 The two principles

The *continuation closure* `PPC` is the least relation that contains every seed $(p \cdot u, p \cdot v)$ with both sides admissible and is closed under appending the same event to both sides while both stay admissible. Under prefix closure it is exactly the set of admissible whiskered pairs (`ppc_iff_whisk`).

Theorem 5.1 (The closure principle; `closure_exact`). *If `Adm` is closed under prefixes, then invariance holds if and only if `sem` agrees, up to $\approx$, on every pair in the continuation closure.*

Proof idea. Every pair in the continuation closure is an admissible whiskered pair (`ppc_whisk`), hence a single generating step of `PEq`, so invariance gives agreement on the closure. Conversely, the proof is an induction on the generated equivalence, using that $\approx$ is an equivalence; the only real case is one whiskered step $p \cdot u \cdot w \sim p \cdot v \cdot w$ with both ends admissible, which lies in the closure by `ppc_iff_whisk`. That lemma is where prefix closure is used: it builds the pair from the seed

by induction on the continuation w , appending one event at a time, and each intermediate word $p \cdot u \cdot w'$ or $p \cdot v \cdot w'$ is admissible because it is a prefix of an admissible word.

Invariance always implies agreement on the seeds themselves (**seeds_necessary**), but the seeds alone are not enough. A hidden register makes the gap concrete (**hw_seeds_insufficient**): events write a register the observation does not show, and a later event reveals it. Every swap seed is then observationally equal at its point, yet invariance fails; the separating pair is a seed continued by the reveal (**hw_closure_witness**). The next principle says when the seeds do suffice.

Theorem 5.2 (The local principle; **local_exact**). *Let Q be a predicate on words that holds on both sides of every seed and is kept by admissible extension, and suppose the semantics respects the future on Q : whenever t and t' satisfy Q , $t \cdot e$ and $t' \cdot e$ are admissible, and $\text{sem}(t) \approx \text{sem}(t')$, then $\text{sem}(t \cdot e) \approx \text{sem}(t' \cdot e)$. If Adm is closed under prefixes, invariance holds if and only if every generator pair is observationally equal at its point: $\text{sem}(p \cdot u) \approx \text{sem}(p \cdot v)$ whenever $\text{Gen}(p, u, v)$ and both sides are admissible.*

Proof idea. By Theorem 5.1 it suffices to compare agreement on the closure with agreement on the seeds (**local_closure**). Seeds belong to the closure, which gives one direction. For the other, an induction on the closure carries the statement that both sides satisfy Q and have $\approx$ -equal semantics: a seed has it by the hypothesis that Q holds on seeds and by the seed condition, and a continuation step keeps it because Q is kept by admissible extension and the semantics respects the future on Q . Prefix closure is used only through Theorem 5.1.

With Q true everywhere this is **local_exact_all**, and the future-respecting hypothesis follows when the observation is compatible with the steps (**resp_of_compat**). In the hidden register it is exactly that hypothesis that fails (**hw_not_resp**). Both principles are elementary facts about congruences generated by local pairs, in the style of semigroup presentations [14]; the work is in fitting every delivery model to them.

State descent is not a generator. It runs one word from two states, s and $N(s)$, so it is a relation on states, not a pair of parallel paths from one point. It enters through the semantics: under state descent the repaired raw semantics and the governed semantics from $N(s_0)$ are the same function on words, so they pose the same invariance question for every presentation (**state_side_transfer**; Lemma 3.5 is the special case of equality on histories). The proof is the same rewrite as for Lemma 3.5: $N(\text{run}(t, s_0)) = \text{grun}(t, N(s_0))$ for every word t (**normalization_descent**). Paths quotient histories; states quotient by repair.

5.3 The history results as instances

Each exact history result of the map is recovered from the local or the closure principle with its original statement, over the original module’s own predicates (Table 1).

Three of these are not trace monoids. At-least-once delivery identifies a word with a shorter one, the live switch has typed phases with labels that change at the switch, and the barrier case needs the closure principle because the migration does not respect the future of the old configuration’s steps. The live switch is the instance where Q matters: the migration does not respect the old configuration’s future, but every seed ends past the switch, where the new configuration’s steps respect its observation.

The cost of fitting a model. The generic theorems are short; fitting a delivery model to them is not. Each recovery has to show that the model’s notion of “same delivery” is exactly the generated equivalence of its presentation, an adequacy argument about words that does not involve the

Delivery model	Recovered as	Principle	Admissible words; generators
Causal delivery	<code>PresentedExecution.causal_exact_recovered</code>	local	causally consistent words; adjacent swaps of concurrent events at a causal point
Declared independence	<code>tconv_exact_recovered</code>	local	causal histories; swaps of declared pairs (the trace monoid)
At-least-once	<code>causal_alo_idem_recovered, alo_exact_recovered</code>	local	at-least-once deliveries, causal (first name) or in any order (second); concurrent swaps and the landing $[a; a] \sim [a]$ after a first delivery
Live switch	<code>det_live_recovered</code>	local, $Q =$ the past switch	typed phases; switch squares and post-switch swaps
Barrier, modulo a migration	<code>amodm_closure_recovered</code>	closure	every word; adjacent swaps (the migration is not compatible with the steps)

Table 1: Exact history results recovered from the two principles. Each recovered theorem states the original’s conclusion over the original’s predicates; the proofs go through Theorems 5.1 and 5.2.

semantics. For at-least-once delivery that argument is about 250 lines of word combinatorics. The recoveries therefore compress the history results into one theorem pair plus a per-model adequacy cost, and the cost is part of the claim (Appendix C).

Not an instance. The single-registry theorem’s commutation condition CC1 is not a row of this table. Its ambiguity is a peak of a rewrite system in which repair steps interleave with events, not a pair of event words. Section 9 combines the local principle on free delivery with a classification of those peaks to derive it.

6 Layer C: long-run behavior

Layer C asks whether runs reach a consistent state and which one. It is the scale of Section 3.4, where repair may never stop: networks of registries that feed each other around a cycle, resolvers that iterate to a fixed point, schedules that interleave updates indefinitely. This section states one kernel for that scale, recovers the settlement results of the map through it, and classifies every way a run was found to fail in one vocabulary.

6.1 The kernel

A system has states X with decidable equality, a list of actions J , and an update $u_j : X \rightarrow X$ for each action, applied asynchronously: a schedule picks one action per step, and it is fair when every action is picked infinitely often. A *fair lasso* from s_0 is a word p that reaches a state x , followed by a nonempty word q that returns x to itself and picks every action; its loop states are the states q visits. Let $N : X \rightarrow C$ be an observation into a set C with decidable equality, and c a target.

Theorem 6.1 (C1; C1_closed, C1_nn). *On a finite X , every fair run from s_0 is eventually always in $N^{-1}(c)$ if and only if no fair lasso from s_0 has a loop state outside $N^{-1}(c)$. The equivalence is constructive when the fiber is closed under every update (Theorem 3.8), and holds for every fiber in double-negated form.*

Proof idea. For the refutation direction, a bad fair lasso (p, q) gives the schedule $p q q q \dots$, which is fair because q picks every action (`lasso_fair`) and visits the bad loop state once per round of

q (`lasso_visits`); no finiteness is used (`bad_lasso_refutes`). For the converse, cut a fair run into consecutive windows that each pick every action (`fair_window`). Since X is finite, two of any $|X| + 1$ window boundaries carry the same state, the segment between them is a fair lasso, and its base point lies in the fiber because no fair lasso is bad (`hits_aux`, `loop_good`); so the run enters the fiber after every time (`nobad_hits`), and a closed fiber keeps it there (`closed_stays`). Without closure, the same lasso argument shows that each state outside the fiber is visited only finitely often, in double-negated form, since whether a state recurs along an arbitrary schedule is not decidable; these statements are combined over the finite list of states, using decidable equality on C to decide membership in the fiber (`state_finitely_often`, `nn_bound`).

Theorem 6.2 (`C2`; `C2`, `C2_loops`). *On a finite X , every fair run from s_0 eventually stays at a state fixed by every update if and only if every fair lasso from s_0 has a single loop state, and that state is fixed by every update.*

Proof idea. The closed form of `C1` and `C2` are instances of one lemma for a decidable set of states closed under every update (`recurrence_closed_exact`). For `C2` the set is the settled states: they are closed (`settled_closed`) and decidable, by decidable equality on X and the finite list of actions (`settled_dec`), so `C2` needs no double negation. The loop form follows from it: a fair lasso whose base point is unsettled is bad, with the base point as its loop state (`onloop_base`), and a settled base point is fixed by every word, so every loop state equals it (`settled_wrun`).

`C1` asks whether runs settle on the right result; `C2` asks whether they settle at all. The refutation direction needs no finiteness: a bad fair lasso refutes settling on any state space (`C1_refutes`, `bad_lasso_refutes`). This is the classical characterization of fair behavior by reachable accepting cycles [15, 10, 16], and it is not claimed as new. The question this section answers is whether it compresses the development’s settlement results.

6.2 The settlement results as short bridges

Each settlement result of the map was recovered by proving that no bad fair lasso exists and reading settlement off `C1` or `C2`. The no-bad-lasso argument for each is between 5 and 11 lines; the whole bridges, with their bindings and assembly, run from 3 lines (fair flush) to about 75 (signed settlement). They share a regime-independent toolkit of about 60 lines: pigeonhole on finite runs, inflationary loops, and a floor lemma for loops below a fixed point (Appendix C).

- *Chaotic iteration reaches the least fixed point* (`chaotic_reaches_lfp_lasso`): a fair loop at a reachable state is inflationary, so it is a single fixed point, which is the least one.
- *Signed networks settle* (`signed_settlement_lasso`): the run from the bottom under the loop’s word stabilizes at a fixed point that lies above the least fixed point and below the loop state, so from sound starts every fair loop is a single fixed point.
- *and settle on the right fixed point* (`signed_fidelity_lasso`): the run from the top bounds every loop by a fixed point, which is unique, so every recurrent class is that fixed point.
- *Fair flush* (`fairflush_lasso`, `fairflushR_lasso`): the originals are `C2`.

Proof idea for signed settlement. `signed_settlement_lasso` keeps the statement of the original and adds that the state type is listed; the switching hypothesis makes every repair monotone in the switched order (`switched_monotone`). The key step is `sr_floor`, an application of the toolkit lemma `loop_floor` to a fair loop at z with word q : the iterates of q from the bottom state are inflationary and stay below z by monotonicity, two of them coincide because the state type is finite (pigeon), and an inflationary loop fixes every action it uses (`infl_loop_fixed`); since q picks every

action, that iterate is a fixed point, so the least fixed point lies below z . From a start below the least fixed point every reachable state stays below it (`wrun_below`), so every fair loop is the least fixed point alone (`sr_below_nobad`), and settlement there is read off the closed form of C1, whose fiber is a single fixed point. From a sound start the run is inflationary, so a fair loop word fixes its base point (`sr_sound_nobad`), which is C2; the remaining conjuncts are the original’s kernel lemmas applied to this settlement.

A cycle that is not fair does not refute anything. The development has a bad cycle that never picks one of the actions, in a network where every fair run settles (`nonfair_cycle_not_refuting`).

6.3 How runs fail

Every long-run obstruction found in the map is a fair lasso, and C1 and C2 tell the kinds apart (`lasso_classification`).

- *Oscillation.* A negation cycle is a bad fair lasso with more than one loop state; it is bad for every target.
- *Livelock.* A bad fair lasso exists although the repair has exactly one fixed point (`flip2_fair_lasso`); this is the B to C interface’s counterexample of Section 8.
- *Settling on the wrong state.* A single settled loop state in the wrong fiber: C2 holds and C1 fails (`copyback_ghost_lasso`).
- *Several fixed points.* Several reachable single-state loops: C2 holds and every target fails C1.
- *Semantic oscillation.* A fair raw oscillation $a \rightarrow b \rightarrow a$ with $N(a) = N(b)$: C2 fails and C1 holds (`semantic_oscillation`). The run never stops moving, but its repaired result never changes.

A good monotone resolver has no bad fair lasso from any start. The last kind is why C1 and C2 are separate conditions: a system can converge in the sense that matters, its result after repair, while never quiescing.

6.4 Boundaries

The recoveries are of finite fragments. Every bridge assumes a finite, listed state type, while the originals assume only finite height, which allows infinitely many states; the chaotic-iteration bridge also needs a finite list of components. The strong form of C1 is classical in general: on a four-state instance with no bad fair lasso, strong settlement implies the limited principle of omniscience (`lpo_leak`). The recoveries avoid this because their targets are fixed points, whose fibers are closed under the steps. Extending the kernel to finite-height state spaces with infinitely many states is open.

7 Layer B: consistent states

Layer B asks which states are consistent. It is a static question about a network’s equations, not about any run: when registries pass values along transport maps, a consistent assignment is one that every map respects. This section shows that the question is a finite-domain constraint satisfaction problem, that the fixed points of the repair dynamics are exactly its solutions, and that the development’s results sit at known points of the algebraic classification of such problems. Little of the mathematics here is new; the section’s role is to place consistency where a mature theory applies, and to connect it to settlement through the interface of Section 8.

7.1 Consistency as constraint satisfaction

Fix a finite family F of transport maps on a finite set D . The *template* Γ_F consists of the graphs $\{(x, f(x))\}$ of the maps in F , together with the unary relations $\{c\}$ for values that some vertices are pinned to. A network whose edges are labeled by maps in F , with some vertices pinned, is then an instance of $\text{CSP}(\Gamma_F)$.

Theorem 7.1 (`section_iff_csp`, `edge_fixed_iff_csp`). *An assignment of values to the vertices of such a network is a section, a consistent assignment that every edge's map respects, if and only if it is a solution of the corresponding instance of $\text{CSP}(\Gamma_F)$. Moreover, for a network without pinned vertices on a store with the usual read and write laws, the states fixed by every edge repair of the network's dynamics are exactly those solutions.*

Proof idea. The correspondence is unfolding: the constraint of an edge $u \rightarrow v$ labeled f holds at an assignment s exactly when $f(s_u) = s_v$ (`csat_econ`), and a pin constraint exactly when the pinned value is taken (`csat_pcon`), so sections agreeing with the pins are the solutions (`section_solution`); each constraint uses the graph of its edge's label or a pin, so the instance is over Γ_F . Finiteness of F and D is not used. For the dynamics, the repair $s[v := f(s_u)]$ fixes s exactly when $f(s_u) = s_v$: reading v after the write gives one direction, and writing back the value just read leaves the store unchanged for the other (`erep_fixed_iff`). Taking every edge gives `edge_fixed_iff_section`, and the correspondence turns sections into solutions.

Merges in which a vertex reads several inputs are graphs of k -ary maps and fit the same way (`hsection_iff_csp`). The identification of global sections with constraint solutions is known in general, from the sheaf-theoretic treatment of contextuality [17, 18]; what is proved here is its form for transport networks, together with the link to the dynamics.

An operation is a *polymorphism* of the graph of f exactly when it commutes with f (`pol_iff_commute`). So the polymorphisms of Γ_F are the operations that commute with every map in F and fix every pinned value (`pol_gamma_iff`), the centralizer of the family in the sense of clone theory. By the dichotomy theorem for finite-domain constraint problems [19, 20], cited and not mechanized, section existence for each fixed finite family is either polynomial or NP-complete, and the polymorphisms decide which.

Proof idea. Applied to k rows $(x_i, f(x_i))$ of the graph of f , an operation p that preserves the graph returns a pair of the graph, which says $f(p(x_1, \dots, x_k)) = p(f(x_1), \dots, f(x_k))$; conversely, commutation sends the column of left entries to a value whose image under f is p of the column of right entries (`pol_iff_commute`). For a pin $\{c\}$ the only rows are constant, so p preserves it exactly when $p(c, \dots, c) = c$ (`pol_pin_iff`), and `pol_gamma_iff` is the conjunction over the relations of the template.

7.2 Where the development's results sit

Groups. For group labels acting by translation, $xy^{-1}z$ is a Mal'tsev polymorphism (`group_maltsev`, `group_pol`), the tractable case of Bulatov and Dalmau [21]; the polynomial criterion mechanized for it is a coboundary test (`section_iff_coboundary`).

Monotone maps on a chain: tractable. A monotone map on a chain commutes with $\min$, $\max$ and the median (`chain_pol`). For pinned networks of monotone maps on a finite chain, arc consistency decides whether a section exists, and the minimum of each arc-consistent domain is one (`ac_exact`). This is the binary, min-closed case of a classical result: max-closed constraints [22], and more generally any semilattice polymorphism [23], make arc consistency decide the instance.

Both hypotheses are needed. On the four-element diamond a monotone map need not commute with join (`diamond_join_fails`), and without monotonicity arc consistency accepts an instance that has no section (`ac_needs_mono`).

Proof idea. The procedure revises every domain at once, keeping a value only if it has a support along every edge into and out of its vertex, and stops when a pass removes nothing; every other pass shrinks the total size of the domains, which starts at most the number of vertices times the size of the chain, so it stops (`measure_nxt`, `ac_run_some`). Revision never removes the value of a section (`sol_ok`, `run_spec`), so if a section exists every final domain is nonempty; this direction does not use monotonicity. Conversely, take the minimum of each final domain (`ac_min_section`). For an edge $u \rightarrow v$ labeled f with minima a and b , the support of a along the edge gives $f(a)$ in the domain of v , so $b \leq f(a)$; the support of b gives some a' in the domain of u with $f(a') = b$, and $a \leq a'$ with monotonicity of f gives $f(a) \leq b$. The pins hold because each domain starts inside its pinned value.

Monotone maps off the chain: hard. Read as two flags, the diamond carries six monotone maps (read one flag, their AND or OR, or a constant) whose only polymorphisms are projections, at every arity (`pol_dfam_iff`); so they have no Siggers, majority or Mal'tsev polymorphism (`dfam_no_siggers`). A linear reduction from 3-SAT to networks over these maps shows section existence NP-complete (`dnet_iff_sat`), for every family that contains them (`mono_family_hard`), and each of the six is needed (`dfam_minimal`). This cell recovers known boundaries: the six maps encode the two-element lattice, whose equation problem is NP-complete [24], and for three-element ordered sets the boundary follows from the classification of unary algebras on at most three elements [25]. What separates the two cells is join preservation. Maps that preserve join, the shape of state-based CRDTs, keep join as a semilattice polymorphism on any lattice (`semilattice_pol`, `joinhom_pol`); monotonicity alone is not enough off the chain.

Proof idea for `pol_dfam_iff`. Projections are polymorphisms of every template of graphs and pins (`proj_pol`). Conversely, by `pol_gamma_iff` a polymorphism p commutes with the six maps. Commuting with the maps that copy one flag into both shows that p sends tuples from the chain $\{(0,0), (1,1)\}$ into that chain, which defines a Boolean function σ of k arguments, and that each flag of $p(x_1, \dots, x_k)$ is σ applied to that flag of the arguments (`p_btup`, `sig_bit1`, `sig_bit2`); commuting with the AND and OR maps makes σ preserve $\wedge$ and $\vee$, and the two constants make it fix 0 and 1 (`sig_comb`, `sig_const`). Such a σ is a coordinate (`bool_hom_proj`): writing each argument tuple g as the join of its unit vectors gives $\sigma(g) = \bigvee_j (g_j \wedge \sigma(e_j))$, so $\sigma(1, \dots, 1) = 1$ yields an i with $\sigma(e_i) = 1$, and if $g_i = 0$ then $\sigma(g) = \sigma(g) \wedge \sigma(e_i) = \sigma(g \wedge e_i) = \sigma(0, \dots, 0) = 0$. So both flags of p are read from argument i , and p is the projection to it.

The reduction's own family. The 3-SAT networks used for the lossy-network hardness result use one fixed family of 26 maps, so constraint satisfaction over that single template is NP-complete (`hard_family_csp`).

Majority off the chain. A family on pairs of Booleans that includes a lossy map commutes with the coordinatewise median but not with meet or join (`median_family`); tractability then follows from the bounded-width results [23, 26], cited, not from a mechanized algorithm.

7.3 What is open

Whether every finite ordered set with at least three elements that is not a chain has only projections as polymorphisms of its monotone maps is open in general as far as the literature checked shows.

The bound on size is needed: on the two-element antichain every self-map is monotone, and the majority and minority operations commute with all of them. The diamond is proved here; an argument for bounded ordered sets is recorded in the development but not mechanized; three- and four-element cases can be read off published classifications of centralizing monoids; and ordered sets without a top and a bottom, from five elements up, are not settled by any source found.

8 The interfaces

The natural objection to a three-layer framework is that executions (A), consistent states (B) and long-run behavior (C) are three classical theories placed side by side. This section answers it with the three interfaces between the layers, each proved without changing the layers, and each with the counterexample that marks where it stops (Fig. 2).

8.1 A to C: Newman as the bridge, and where fairness falls short

In layer C's terms a reduction is a step of some action that changes the state, and its normal forms are the settled states (`nf_iff_stable`). Under termination from the start and decidable normality, local joinability at reachable peaks, confluence and unique normal forms coincide (`newman_bridge`); the converses need no termination (`cr_peaks`, `cr_unf`). This is Newman's lemma [27] restricted to the reachable states. The single-registry theorem has the same shape, with termination of repair supplying the hypothesis and CC2 being state descent, but it is not derived as an instance of `newman_bridge`: its derivation joins peaks of the configuration rewrite system through the canonical-execution kernel (`classified_peak_exact`, `nd_confluent_from`; Section 9).

Proof idea. In `newman_bridge`, local joinability at reachable peaks gives confluence by Newman's lemma applied to the reachable states, which are closed under steps and inherit termination (`peak_exact`, `newman_on`, `SN_star_closed`). Confluence gives local joinability at once, and unique normal forms because two normal forms below a state join and a normal form reduces only to itself (`cr_unf`); neither direction uses termination. For the remaining direction, termination and decidable normality give every reachable state a normal form below it, by well-founded induction in which decidability chooses between stopping and taking a step (`sn_reach_nf`). Two reachable states then have normal forms that both lie below the start, so unique normal forms makes them equal, and that normal form joins the two states.

Fair settlement does not replace termination. In the standard four-point system $k_a \leftarrow k_b \leftrightarrow k_c \rightarrow k_d$, every peak joins and every fair run settles, yet the run that only swaps k_b and k_c is an infinite reduction that starves the other actions, and k_a and k_d are two normal forms below k_b (`four_point_fair_not_confluent`).

Theorem 8.1 (`fair_starves`, `fair_sn_exact`). *If every fair run settles (C2), then every reachable loop of moving steps starves some action: some action it never picks would move every state on the loop. Consequently, on a finite state space, termination holds exactly when no reachable loop starves an action, and then Newman's lemma applies (`fair_newman`).*

Proof idea. For `fair_starves`, take a reachable loop of moving steps at x with word q . Each action occurs in q , or fixes some state on the loop, or moves every state on the loop and is starved; equality on X is decidable, so the cases can be decided. If no action is starved, insert before each step of q every action that fixes the current state; this changes no state (`wrun_ins`) and makes the word pick every action (`in_ins_orig`, `in_ins_fixer`), so it is a fair lasso whose base point x is

moved by the loop's opening step, a bad lasso that refutes C2 (`bad_lasso_refutes`). No finiteness is used in this part. Finiteness enters in `sn_iff_no_loop`: a reduction chain from the start with as many steps as there are states repeats a state (`pigeon_aux`), and the segment between the repeats is a reachable moving loop, so without such loops chains are bounded and reduction terminates (`acc_bound`), while a terminating relation has no cycle (`sn_no_cycle`). `fair_sn_exact` combines the two, and `fair_newman` applies `newman_bridge`, with normality decided by checking the finitely many actions (`lstep_dec`).

Fair settlement replaces termination exactly when reductions cannot be hidden forever by starving an action. The condition is exact for termination, not for confluence: a network with a starved loop can still be confluent (`drop_starved_confluent`). Whether Theorem 8.1 is stated in the fair-termination literature [28, 29] is open: we did not find it there; the statement is close to the standard picture, in which an infinite run is either fair or starves some action forever.

8.2 B to C: the fixed points are the solutions

On a store with the usual read and write laws, a state is fixed by every edge repair $s[v := f(s_u)]$ if and only if it is a section, if and only if it solves the network's constraint problem (`edge_fixed_iff_section`, `edge_fixed_iff_csp`); the same holds for resolver repairs that recompute a vertex from several inputs (`rupd_fixed_iff_csp`). Read in layer C:

Theorem 8.2 (`singleton_iff_solution`, `settled_canon_exact`). *The fair recurrent classes that are single states are exactly the reachable solutions of the constraint problem. If every fair run settles (C2), then every fair run settles on the result c if and only if every reachable solution lies in the fiber of c .*

Proof idea. For `singleton_iff_solution`, suppose a fair loop at x visits only x . For each action j pick a position of the loop word where j occurs; the states just before and just after it are both x (the later one is a loop state, or, at the last position, the loop's return to x), so $u_j(x) = x$. Conversely, a reachable fixed point carries the loop that applies every action once, which is fair, returns to it and visits nothing else (`settled_wrun`); this uses a nonempty list of actions. For `settled_canon_exact`, the lasso schedule of such a single-state loop is fair and visits the solution infinitely often (`lasso_visits`), so settling on c puts every reachable solution in the fiber of c (`canon_needs_solutions`). Conversely, under C2 every fair run is eventually always at settled states, each reached by a word and so a reachable solution, hence eventually always in the fiber. Neither direction uses finiteness.

So a state that settles on the wrong result is a reachable solution outside the target fiber (`ghost_refutes`), and two reachable solutions with different results refute every target (`multistable`); a network with no solution has no raw settlement (`no_solution_no_settlement`). The converse direction fails in the way that matters most:

Proposition 8.3 (`flip2_unique_solution_livelock`). *There is a one-vertex network whose constraint problem has exactly one solution and whose repair dynamics has a fair run that never settles.*

Proof idea. The proof is a computation on one instance. The vertex takes values z, a, b and has two self-loop edges: the swap, which fixes z and exchanges a and b , and the drop, which sends a to z and fixes b and z . The swap edge forces the value z , which the drop fixes, so z is the only section and, by `edge_fixed_iff_section`, the only state fixed by both repairs. The word swap, drop, swap takes a to b , leaves b in place and returns to a ; it picks both actions and its base point a is not fixed, so it is a bad fair lasso and C2 fails (`bad_lasso_refutes`).

A unique consistent state does not guarantee that runs reach it. This is the distinction prediction P2 (Section 10.2) bet on. One leak is recorded: a network with no solution can still settle canonically, through a semantic oscillation whose repaired result never changes (`no_solution_canon_settles`).

8.3 A to B: consistency needs to hold only after repair

A per-vertex family of maps that commutes with every edge map sends sections to sections (`natural_maps_sections`). For one network the exact condition is commutation on the values sections take (`preserves_exact`), and the commuting squares are necessary edge by edge for edges that are not self-loops (`square_necessary`), with counterexamples to the naive converse (`natural_converse_fails`). A uniform family is a unary polymorphism (`uniform_natural_is_unary_pol`).

For events in a network of registries, strict commutation with the repair is too strong. The condition that matters is descent: an event must respect consistency only after repair.

Theorem 8.4 (`xu_is_descends`, `candidate_implies_xu`, `xu_iff_respects`). *The event condition XU of the federation paper [6] is state descent along each registry’s repair on its valid states. Strict commutation implies it, and it holds exactly when each event passes to the quotient of the valid states by the repair.*

Strict commutation is sufficient but not necessary. On a supply federation XU holds, strict commutation fails, and every order of events converges (`strict_not_necessary`).

Proof idea. `xu_is_descends` unfolds the definition of XU. The rest are facts about an operation e and a map N on a set P , used with N the repair of the event’s registry and P its valid states: when N maps P into itself and is idempotent there, descent $N(e(Nx)) = N(e(x))$ is equivalent to respecting the kernel of N (`descends_iff_respects`), and strict commutation implies descent, since $N(e(Nx)) = e(N(Nx)) = e(Nx) = N(e(x))$ (`commutes_descends`). For the federation, closure and idempotence are its assumptions that repair preserves validity and that a later repair absorbs an earlier one (`fed_closed`, `fed_idem`). `strict_not_necessary` is computed on a two-registry supply federation (`supply_instance`): the repair overwrites the supplier’s listed flag from the manufacturer’s recall flag and keeps the count, and no supplier event reads the flag, so descent holds, while unlisting and then repairing restores the flag and repairing and then unlisting leaves it cleared, so strict commutation fails. Convergence of every order follows from the federation theorem through `xu_implies_c1_c2` and `fed_permutations_converge`.

9 The compensation-commutativity theorem as the base case

The single-registry theorem of Section 3.3 was proved directly, before the framework existed. This section derives it from the framework’s kernels, used unchanged. It is a derivation, not a prediction: the framework was built after the theorem, with it in view. What it shows is that the kernels, as they are, reproduce the exact statement, and that each of its conditions is a named condition of a layer. The setting is Theorem 3.6’s: free delivery, terminating repair, and $N = \rho^*$.

The dictionary.

- *CC2 is state descent.* From a start s_0 , every event descends along N on the states reachable from s_0 if and only if CC2 holds there (`nd_descends_cc2`); globally this is Proposition 3.3.

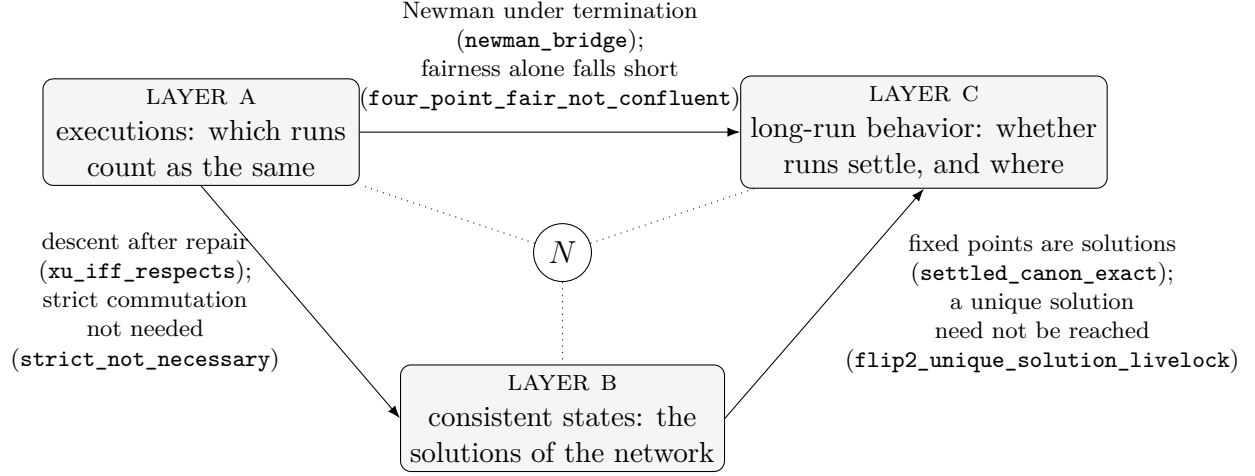


Figure 2: The three layers, each reading the repair map N , and the three interfaces of Section 8, each labeled by its theorem and the counterexample that marks where it stops.

- *CC1 is history descent on free delivery.* The governed semantics from the repaired start, $t \mapsto \text{grun}(t, N(s_0))$, is invariant under the free-delivery presentation, whose generated equivalence is equality up to permutation (`swap_peq_perm`). By the local principle (Theorem 5.2) this holds exactly when every adjacent swap commutes after every word (`nd_history_seeds`), and given state descent it is CC1 on the states reachable from s_0 (`nd_history_cc1`). History descent alone gives CC1 only at the repaired states the governed runs reach; state descent carries it to the raw reachable states, where it is needed because an event can fire before repair.
- *Termination is the progress side*, supplied by the potential Φ .
- *Newman joins them.* A peak between an event and a repair step is joined by state descent, and a peak between two events by CC1 (`nd_state_peaks_join`, `nd_history_peaks_join`); the peak classification of the canonical-execution kernel (`classified_peak_exact`) then gives confluence (`nd_confluent_from`).
- *The converse is by the same small runs* as in Section 3.3 (`nd_tiny_cc1`, `nd_tiny_cc2`).

Proof idea for the dictionary. For `nd_descends_cc2`, one repair does not change the repaired form of an invalid state, $N(\rho(\sigma)) = N(\sigma)$ (`nd_N_rho`, because runs from an empty buffer are deterministic); descent at the reachable states σ and $\rho(\sigma)$ then gives CC2 at σ . Conversely, CC2 keeps $\text{gov}(e)$ unchanged along the repair chain from a reachable x to $N(x)$, by induction on the chain (`nd_cc2_chain`), which is descent at x . For `nd_history_seeds`, the future-respecting hypothesis of Theorem 5.2 holds trivially, since the governed semantics of $t \cdot e$ is $\text{gov}(e)$ applied to that of t , so `local_exact_all` reduces invariance to the adjacent swaps. For `nd_history_cc1`, state descent makes $N(\sigma)$ a governed run from $N(s_0)$ for every reachable σ (`nd_reach_canon`) and gives $\text{gov}(e)(\sigma) = \text{gov}(e)(N(\sigma))$, so the swap after that run is CC1 at σ ; the converse applies CC1 at the governed runs, which are reachable (`nd_reach_grun`). The peaks join directly: an event against a repair step meets at $(\text{gov}(e)(s), B - e)$ by CC2, and two events meet at $(\text{gov}(e_2)(\text{gov}(e_1)(s)), B - e_1 - e_2)$ by CC1.

Theorem 9.1 (`nd_derived_from`, `nd_cc_exact`). *For every start s_0 , every buffer delivered from s_0 has a unique normal form if and only if free-history descent holds from $N(s_0)$ and every event*

descends on the states reachable from s_0 . The three original statements of the single-registry theorem follow word for word.

Proof idea. For sufficiency, `nd_history_cc1` turns the two descent conditions into CC1 on the states reachable from s_0 , and `nd_confluent_from` gives confluence from (s_0, B) : `classified_peak_exact` is Newman’s lemma on the reachable configurations, with termination from Φ and a classifier under which every nontrivial peak is a state peak or a history peak (`nd_cls_complete`). Confluence gives unique normal forms. For necessity, `nd_tiny_cc1` and `nd_tiny_cc2` run a word that reaches σ and then the two small buffers; both ends are normal forms because ρ^* reaches a valid state and the buffer is empty, so unique normal forms make them equal and give CC1 and CC2 at σ . The dictionary turns these back into the two descent conditions. The original statements follow by rewriting with the dictionary (`nd_dictionary_from`); `nd_cc_exact`, for configurations over any reachable state, applies the start-relative form at that state and uses transitivity of reachability.

Only one proof in the file cites the original theorems: `nd_derived_matches_original`, which shows that the derived statements are the originals’ by the dictionary alone. Every other theorem of the file remains closed under the global context when the original theorems are turned into admitted axioms in a scratch copy, so none of them depends on the originals. A small instance where both sides hold shows the hypotheses are satisfiable (`nd_nonvacuous`).

What it shows and what it does not. The base case shows that the framework reproduces the theory’s central theorem with its conditions named by layer: state descent at one event, history descent over sequences, termination as progress, Newman as the bridge. It does not show reach, because the framework was designed with this theorem in view; that is what Section 10 tests.

10 Testing the framework

The framework of Sections 5 to 8 was built after the results it organizes were known. Recovering those results, including the single-registry theorem (Section 9), shows that the framework *compresses* them; it does not show that the framework reaches anything it was not built from, and it predicts nothing. This section reports two stricter tests: recovering classical theorems from outside the development with the framework frozen (Section 10.1), and predictions committed before they were tested (Section 10.2). It also reports what failed.

10.1 Outside theorems, framework frozen

Protocol. For each target theorem, every existing proof file was frozen, the pass criterion was written down before the proof was attempted, and anything the frozen framework could not express was recorded as a *leak* rather than repaired. The work for each target is one new file. A target passes when its statement follows by a short bridge from the frozen kernels, with the target-specific work confined to the target.

Dijkstra’s self-stabilization (layer C; passed). For a finite system and a legitimate set L closed under every step, every fair run from s_0 eventually stays in L if and only if no fair lasso from s_0 has a loop state outside L (`stab_fair_from`). The proof is one line: Theorem 3.8 with the observation “is the state in L ”. Dijkstra’s K -state token ring [30] then stabilizes for every ring of $n \geq 2$ machines with $K \geq n$, under fair schedules (`ring_fair_selfstab`) and under a central daemon that picks any enabled move (`ring_unfair_selfstab`), Dijkstra’s own setting; the

Target	Layer	Outcome	Leaks
Self-stabilization	C	passed	listed finite state type; an adapter for systems that never quiesce; no simultaneous moves
Invariant confluence	B	passed	partition model for necessity; item-wise merges only
CRDT SEC	A	partial	merge of two replica states

Table 2: Outside theorems recovered with every existing file frozen. Bridge sizes, target-specific work and leaks are recorded in the development (THEORY.md, “Testing the framework”).

ring-specific work is his fresh-value argument. With $n = 4$ and $K = 2$ a bad fair lasso exists (`ring_bound_matters`). Dijkstra’s manuscript also notes that $K \geq n - 1$ suffices under a central daemon [31], later proved sharp [32]; that bound is not mechanized here. The ring has been mechanized before, in PVS [33] and in Coq [34]; the point of this test is the transport, not the mechanization. Under an arbitrary daemon, the exact condition is fair stabilization together with the absence of a reachable loop outside L that starves a machine (`unfair_exact`), a consequence of the A to C interface whose relation to the fair-termination literature [28, 29] is stated as open.

Invariant confluence (layer B; passed). The results of Section 4.3 were obtained in this protocol: the polymorphism bridge (`pol_iconfluent`) is short, the exact form with reachable states (`iconf_iff_pol_reach`) and the counterexample (`uq_not_pol`) stay inside the new file, and in a replica model the protocol in which a server commits a transaction only if its local result satisfies I and merges every state it receives keeps every server I -valid in every execution if and only if the transactions are invariant confluent (`replica_exact`).

CRDT strong eventual consistency (layer A; partial). For op-based CRDTs, with the identity repair, strong eventual consistency from s_0 holds exactly when concurrent operations commute wherever they can be delivered causally (`opsec_exact`). Shapiro et al.’s condition is a corollary, and the convergence theorem of Gomes et al. [35] follows in their own definitions (`gomes_convergence`). For state-based CRDTs, replicas with the same history set have equal states in every execution of Shapiro et al.’s replica model, when every update is unique (`cvrtdt_sec`), but the step that merges one replica’s state into another is not expressible in the frozen kernels and was reduced outside them. This is the weakest of the three tests: CRDT regimes were in the development before the framework was written.

What the leaks share. The kernels follow one state through time. They cannot combine two concurrent states (the merge of two replica states) or two concurrent steps (simultaneous moves), and the A to C interface speaks of settled states, so a system that never quiesces needs an adapter. These are recorded as boundaries of the framework and were not repaired: a later version that addresses them should be judged on outside theorems that were not used to design it.

10.2 Predictions committed before their tests

The freeze and the protocol. On 2026-10-11 the eight kernel files were frozen; their hashes are checked in continuous integration, so the build fails if one changes. Candidates were drafted from the frozen kernels only, each with a literature check and pass and fail criteria. A chosen candidate is committed unchanged, the commit being its timestamp, before any test; a separate run then tries to establish or refute it, and the outcome is recorded whatever it is. Three candidates were drafted,

and P2 is one of them. Another, about routing, was refuted while being prepared for commitment and was never committed (a three-node network with a unique solution in every failure scenario in which stale routes circulate forever on a cycle the destination cannot reach). It is recorded with the others, and the committed claims are reproduced in Appendix E.

P2: a bet on a proved distinction (passed). ¹ The interface between layers B and C contains two results that point opposite ways: a unique consistent state guarantees settling when the repair is monotone (`q1_unique_iff`), and does not in general (`flip2_unique_solution_livelock`; the first needs a top element and the dual step laws). P2 bet that the classical stable marriage market, whose blocking-pair update is not monotone, falls on the second side: some market with at most four men and four women has exactly one stable matching and a cycle that never settles under a schedule that gives every pair a turn. The claim was committed with an estimated probability of 0.6. The nearest published cycling instances [36, 37], the latter read as reproduced by Chen, Skowron and Sorge [38], do not decide it, since both have several stable matchings; Knuth’s cycling example [39] uses a different rule, in which divorced partners marry each other; and random resolution converges [40]. An exhaustive search found the property already at three by three, and one instance is mechanized against the frozen kernels (`p2_pass`, through `bad_lasso_refutes`): the unique stable matching, and a loop that names each of the nine pairs once and passes only through unstable matchings.

n	profiles	with a unique stable matching	of those, with a fair cycle
2	16	14	0
3	46,656	34,080	3,240
4	110,075,314,176	65,867,261,184	17,655,840,000

At every size tested, every profile with two or more stable matchings has a fair cycle. Outside the scored claim, the instance also has cycles that are fair in the stronger sense that every pair that blocks infinitely often is resolved infinitely often. The framework posed the question and supplied the reason for the bet and the refutation tool; it did not compute the answer, which came from the search.

P3: a computed prediction about real software (passed). P3 is of a different kind: a model of real code, built through the framework, computes an answer, and the running software is the experiment. CouchDB keeps a revision tree per document and, after every replicated update, prunes it to a configured depth; pruning is the repair. A document is edited three times and then deleted, the four revisions are delivered in each of the 24 orders, each with its full history, to a fresh document id with depth 3. `gsm` (Section 12), run on the model and reading its result through the at-least-once exact theorem (`alo_exact`: convergence under every order and duplication exactly when the updates commute at reachable states and are idempotent), locates the failure in one pair of revisions after the deletion and shows that duplicates are not its cause; state descent explains why pruning after every update, but not pruning once at the end, changes the outcome. A model transcribed from the source computed the outcome of every order, and three claims were committed before either system was run: CouchDB 3.5.3 brings the deleted document back on exactly the 6 orders in which the first revision arrives last; with per-update pruning switched off and one compaction at the end, on none; and PouchDB 9.0.0, an independent implementation of the same algorithm, on exactly the 12 orders in which the first revision arrives after the deletion. All 216

¹Predictions are numbered as committed in the development; P1 was reserved for the routing candidate, which was refuted before commitment.

scored observations (three claims, 24 orders, three repetitions) matched, with no nondeterminism, and the full leaf sets matched as well.

The resurrection itself is a known, accepted consequence of pruning [41, 42]; the order rule, the healing step that rescues some orders, the effect of the pruning setting, which is documented as a performance switch, and the disagreement between the two implementations were not found in any source.

What failed in P3. A larger scenario with partial histories was run but not scored. There the CouchDB model was wrong on 3,009 of 5,040 orders. The cause was a filter that the model applied to every replicated update and that CouchDB applies only when the database has a validation function (or the request carries a design document or attachments). A corrected model, kept separate and not scored, matches every order in both configurations. The correction exposed a CouchDB behavior: with partial histories, the same revisions in the same order can leave a document live or deleted depending only on whether the database has a validation function, even one that accepts everything. CouchDB’s own replicator reaches it, through an optimization that skips the missing-revisions check after a run of batches in which everything was missing. It was reported to the CouchDB project, a maintainer confirmed it as a bug [43], and a fix was merged upstream [44]. Follow-up tests on CouchDB 3.5.3, archived with the replicator scenarios, narrowed the trigger: it needs databases with different revision limits and a document history deeper than the lower limit, and it occurred at every mismatch tested, 3, 10, 100 and 999 against 1000 (through the replicator with default settings at 100 against 1000). With the same limit on every database (1000, 2, 3, 10 or 100) it never occurred. The re-sent revision is dropped before the validation function runs, so the function’s code plays no part.

What may be claimed. The framework compresses the results it was built from (Section 9); some of its structure transports to theorems outside the development (Table 2); and it has made two correct preregistered predictions, one a bet on which side of a proved distinction a new system falls (P2), one a model computation checked against running software (P3). It does not derive answers by theorem alone, and two predictions are a small sample.

11 The regime map

The results of Sections 3 to 8 are the organizing core of a larger body of work: a map of the regimes of governed concurrent state. In every regime it does not exclude by design, the map records a machine-checked exact condition, a proof that no efficient exact condition exists unless $P = NP$, or an open problem listed in an audit. This section summarizes it (Fig. 3, and by family in Appendix D); the regime-by-regime audit, with the Coq theorem behind each entry, is part of the development.

Exact. Single registries; causal and at-least-once delivery; CRDTs; stream processors; acyclic federations of registries; a distributed propagation model, acyclic and on monotone cycles with or without reset epochs; rootless propagation on every finite invertible network; reconfiguration of a registry or a federation, at a barrier and live, under free, declared, causal and at-least-once delivery; propagation over channels on acyclic networks, and with versioned merging on every network in which each target has one source; and composition of blocks of any engines, including cyclic monotone collapse.

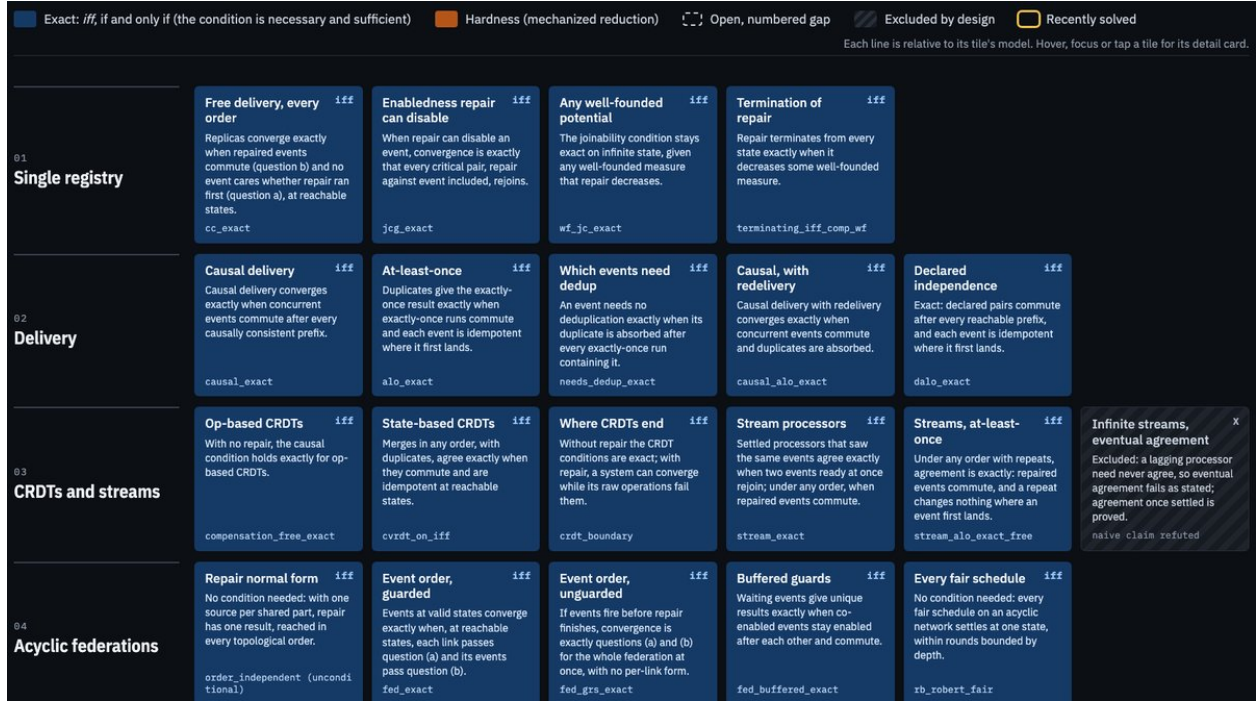


Figure 3: Part of the Convergence Atlas: one tile per regime, grouped by family, each marked exact, hard, open or excluded and linked to its theorem. The full atlas has 60 regimes in 11 families.

Hard. Consistency of a lossy network without a spanning root is NP-complete, by a mechanized reduction from 3-SAT. Minimum coordination is characterized exactly and is NP-hard, from Max-Cut on invertible networks and from 3-SAT on lossy ones.

Counting and gluing. The first cohomology of the nerve is classified and counted over $\mathbb{Z}/2$, and gluing local sections is exact on sub-federation covers; positive assembly of local sections at the level of variables is narrowed but still open.

Open. Seven convergence questions remain: rootless lossy networks, the distributed model off its current hypotheses, rootless dynamics in which edges write values, existence of fixed points in the resolver reading, distributed collapse of a cyclic block, reconfiguration with propagation in flight, and versioned channels on networks with several sources or with cycles. Most concern cycles; the audit states them as one question, what makes composition exact on cycles.

Excluded by design. Least fixed points without the ascending chain condition, Byzantine participants, nondeterministic repair, probabilistic schedules, infinite networks, and real-time semantics.

Completeness of the map. A coverage pass derived the space of regimes from the axes of the model's premises and mapped every meaningful combination to the audit. The combinations no entry covered became open problems or design exclusions; six of the seven open questions came from that pass.

Tiles and gaps. The atlas has nine open tiles for the seven open convergence questions: one question appears in two families, and one tile marks the open sheaf-assembly item above, which is not a convergence question.

The evidence base. The development’s axiom-free gate checks 4,797 named results, every one closed under the global context, with no axioms and no admitted proofs anywhere in the source, checked on three versions of Coq and Rocq on every change. The counts are kept equal to the gate by a script that also checks every theorem name cited on the atlas and the website against the source and the gate.

12 Checking the conditions at build time

`gsm` [45] is a Go library that runs the exact conditions of this paper on concrete systems. A team declares its state as typed variables, its business rules as invariants with a repair, and its events as raw transitions. Building the machine enumerates the finite state space and checks that repair terminates (WFC) and that repaired events commute (CC1 of Theorem 3.6) on every valid state. A `gsm` machine applies each event and its repair as one step, the atomic case of Section 3.3 (“When repair runs”), so CC2 is not needed there (`atomic_exact`). If the checks pass, the build returns a machine that applies an event with one table lookup; if not, it returns no machine and reports a pair of events, a state, and the two orders that diverge. For the shelf of Example 3.2, written with stock from 4 to 8 so that “remove 1” always has room, the build refuses the machine and reports the example’s divergence: from 4 in stock, “add 3” then “remove 1” gives 4, the other order gives 5.

Checked twice. The result of the Go code is not trusted on its own. A checker extracted from the axiom-free Coq development re-certifies the machine’s tables in process, and the build fails closed if it does not. Relative to a trusted base, the extraction mechanism, the glue that calls the generated code, and the translation of `gsm`’s tables into its input, a bug in `gsm`’s own verification cannot hand back a machine that does not converge.

Scale. The state space of a realistic system is a product of variable domains and quickly exceeds what can be enumerated. `gsm` checks it in pieces, each justified by a theorem of the development: by footprint component, the groups of variables that events and rules touch together, which is exact for the whole machine when the rules are combinators; by one item of a keyed collection, which stands for any number of items; by representative values for integer variables that are only compared and copied; and, for networks of registries, by two local checks per link (called C1 and C2 in `gsm` and the earlier papers; they are federation conditions, not the layer C conditions of Section 6) on acyclic networks, with opt-in modes for monotone cycles and for cycles with a named coordination plan. The cost is then governed by the largest piece, not by the product of all domains.

Changes in flight. A rules or schema change made while events are in flight is classified before deployment as safe at any time, safe behind a barrier, unsafe with a replayable witness, or unknown when a search bound is hit, under the delivery model the deployment actually has.

What is not exact. `gsm`’s checks are sufficient forms of the atomic-repair condition at the reachable states: they range over the whole enumerated state space, not only the states a system can reach, so a machine can be refused although it would converge from its actual start. Domains

must be finite, and a single tightly coupled group of variables that is too large to enumerate is out of reach. Every guarantee in a build report cites the theorem behind it, and a test keeps those citations equal to the gated names.

13 Related work

CRDTs and coordination avoidance. CRDTs [1] obtain convergence by designing operations that commute; Section 4 shows they are the case where repair does nothing, that their conditions are exact on reachable states, and that every convergent repaired system is a CRDT on its governed steps. The CALM theorem [46, 47] characterizes the programs with a coordination-free implementation as the monotone ones; the monotone-cycle regime of the map is related, a monotone operator whose least fixed point is reached in any fair order, carrying business invariants. Invariant confluence [2, 11] asks which transactions preserve an invariant without coordination, with the invariant checked and never repaired (Section 4.3).

Invariants on weakly consistent replication. RedBlue consistency [4] labels operations as commuting or globally ordered; Indigo and IPA [5, 48] combine invariant preservation with repair and with reservations, building on escrow [49]. These systems ground convergence in operation commutativity plus invariant preservation, transforming operations where needed. Normalization confluence admits operations that do not commute and that violate the invariant, and proves convergence by Newman’s lemma on the repaired rewrite system: compensation commutativity is local confluence, not operation commutativity. Bayou [50] resolved conflicts by application-specific merge procedures after a tentative order; sagas [3] use compensation to undo partial work. Normalization confluence gives compensation a convergence theory: exactly when repairs make concurrent orders agree.

Rewriting, fixed points and traces. The single-registry theorem is the critical-pair theorem [8, 9] specialized to a system whose critical pairs are normal forms, with Newman’s lemma [27] as the bridge. Chaotic iteration to a least fixed point is classical in abstract interpretation [51] and in discrete dynamics [52]. Layer A is trace theory [12, 13] generalized to admissible words and non-commutation generators, with the congruence facts of semigroup presentations [14].

Constraints and long-run behavior. Layer B places consistency in the algebraic theory of constraint satisfaction [23, 19, 20], and the identification of global sections with constraint solutions follows the sheaf-theoretic treatment of contextuality [17, 18]. Layer C’s kernel is the fair-cycle characterization of model checking [15, 10, 16]; its relation to fair termination [28, 29] is stated as open.

Mechanized convergence. Gomes et al. [35] verified strong eventual consistency for op-based CRDTs in Isabelle; their theorem is recovered here in their own definitions (Section 10.1). Dijkstra’s self-stabilizing ring has been verified in PVS [33] and Coq [34]. The development here differs in scope rather than in any single proof: it mechanizes an exact condition, a hardness result or a listed open problem for every regime of a map that it does not exclude by design, with the counterexamples that fix each hypothesis.

14 Limitations and what is not claimed

- *No new mathematics in the tools.* Descent is compatibility with a kernel; Newman’s lemma, trace theory, constraint-satisfaction algebra and fair cycles are classical and credited where they are used. The contribution is the exact conditions, the map, the mechanization and the framework’s tests, not new technique.
- *The three layers are not one principle.* Descent unifies the single-event conditions; across scales the conditions form a hierarchy with proved separations (Proposition 3.10). Layer B is not a descent condition.
- *Recoveries compress; they do not confirm.* The framework was written after the results it recovers. Outside theorems and two preregistered predictions are the evidence of reach, and two predictions are a small sample.
- *The framework does not derive answers by theorem alone.* P2 was a bet on which side of a proved distinction a system falls; P3 was a model computation checked against running software.
- *Layer C is exact on finite state spaces.* The settlement recoveries are of finite fragments, and the strong form of C1 is classical in general.
- *A unique solution does not give convergence* (Proposition 8.3), and arc consistency does not decide existence without the chain and monotonicity hypotheses.
- *gsm’s static checks are sufficient, not exact.*
- *Scope.* “Exact” is relative to the model each result states. Least fixed points without the ascending chain condition, Byzantine participants, nondeterministic repair, probabilistic schedules, infinite networks and real-time semantics are excluded by design.

15 Conclusion

Repair can replace raw commutativity exactly when the result after repair does not depend on the order. At one event, when events can arrive before repair finishes, this requires state descent; together with history descent over finite histories, and terminating repair, it is exact for a single registry; over infinite runs on finite state spaces, canonical recurrence is exact. The conditions form a proved hierarchy, and counterexamples fix the qualifiers they name. A three-layer framework with proved interfaces organizes them across a map of regimes, recovers the central theorem without circularity, transports to classical theorems from outside its development with its own files frozen, and has made two correct preregistered predictions, one of them about widely used databases.

The open problems that matter most are composition on cycles, where most of the map’s open questions sit; long-run behavior beyond finite state spaces; a way to combine concurrent states or steps in the execution layer, which the CRDT and self-stabilization tests pointed to; and an exact structural condition that decides, for a class of systems, when a unique consistent state is also reached, which would let the framework compute answers where it can now only pose the question.

Artifact availability

The Coq development, the documentation and every script used in this paper are public at <https://github.com/blackwell-systems/normalization-confluence>; the paper cites the state of the repository at tag `paper-v4`. To rebuild and check the proofs:

```
cd coq && make && ./verify.sh
```

`verify.sh` prints the assumptions of every gated result and fails unless each is closed under the global context and their number reaches the gate (4,797). Continuous integration runs the same command on Coq 8.18, Coq 8.20 and Rocq 9.3. The P2 search and its tests are in `research/predictions/p2/`; the P3 models, harnesses and result files are in `research/predictions/route2/`; the replicator scenarios are in `research/couchdb-replicator/`; the committed predictions and their outcomes are in `research/predictions/`. The checker of Section 12 is at [45].

Acknowledgments

Drafts of the text and parts of the Coq development were prepared with AI assistance under the author’s direction; all theorems are machine-checked in Coq, and the author takes responsibility for the content.

References

- [1] M. Shapiro, N. Preguiça, C. Baquero, and M. Zawirski. Conflict-free replicated data types. In *Stabilization, Safety, and Security of Distributed Systems (SSS)*, LNCS 6976, pages 386–400, 2011.
- [2] P. Bailis, A. Fekete, M. J. Franklin, A. Ghodsi, J. M. Hellerstein, and I. Stoica. Coordination avoidance in database systems. *Proceedings of the VLDB Endowment* 8(3):185–196, 2014.
- [3] H. Garcia-Molina and K. Salem. Sagas. In *SIGMOD*, pages 249–259, 1987.
- [4] C. Li, D. Porto, A. Clement, J. Gehrke, N. Preguiça, and R. Rodrigues. Making geo-replicated systems fast as possible, consistent when necessary. In *OSDI*, pages 265–278, 2012.
- [5] V. Balesgas, S. Duarte, C. Ferreira, R. Rodrigues, N. Preguiça, M. Najafzadeh, and M. Shapiro. Putting consistency back into eventual consistency. In *EuroSys*, 2015.
- [6] D. Blackwell. Normalization confluence in federated registry networks. Zenodo, 2026. <https://doi.org/10.5281/zenodo.18677400>.
- [7] D. Blackwell. Normalization confluence for registry-governed stream processing. Zenodo, 2026. <https://doi.org/10.5281/zenodo.18671870>.
- [8] D. E. Knuth and P. B. Bendix. Simple word problems in universal algebras. In *Computational Problems in Abstract Algebra*, pages 263–297. Pergamon, 1970.
- [9] G. Huet. Confluent reductions: abstract properties and applications to term rewriting systems. *Journal of the ACM* 27(4):797–821, 1980.
- [10] E. A. Emerson and C.-L. Lei. Modalities for model checking: branching time logic strikes back. *Science of Computer Programming* 8(3):275–306, 1987.
- [11] M. Whittaker and J. M. Hellerstein. Interactive checks for coordination avoidance. *Proceedings of the VLDB Endowment* 12(1):14–27, 2018.
- [12] P. Cartier and D. Foata. *Problèmes combinatoires de commutation et réarrangements*. Lecture Notes in Mathematics 85, Springer, 1969.

- [13] A. Mazurkiewicz. Concurrent program schemes and their interpretations. DAIMI Report PB-78, Aarhus University, 1977.
- [14] J. M. Howie. *Fundamentals of Semigroup Theory*. Oxford University Press, 1995.
- [15] E. M. Clarke, E. A. Emerson, and A. P. Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Transactions on Programming Languages and Systems* 8(2):244–263, 1986.
- [16] M. Y. Vardi and P. Wolper. An automata-theoretic approach to automatic program verification. In *Logic in Computer Science (LICS)*, pages 332–344, 1986.
- [17] S. Abramsky and A. Brandenburger. The sheaf-theoretic structure of non-locality and contextuality. *New Journal of Physics* 13:113036, 2011.
- [18] S. Abramsky. Contextual semantics: from quantum mechanics to logic, databases, constraints, and complexity. *Bulletin of the EATCS* 113:137–163, 2014.
- [19] A. A. Bulatov. A dichotomy theorem for nonuniform CSPs. In *Foundations of Computer Science (FOCS)*, pages 319–330, 2017.
- [20] D. Zhuk. A proof of the CSP dichotomy conjecture. *Journal of the ACM* 67(5):30:1–30:78, 2020.
- [21] A. Bulatov and V. Dalmau. A simple algorithm for Mal’tsev constraints. *SIAM Journal on Computing* 36(1):16–27, 2006.
- [22] P. Jeavons and M. C. Cooper. Tractable constraints on ordered domains. *Artificial Intelligence* 79(2):327–339, 1995.
- [23] P. Jeavons, D. Cohen, and M. Gyssens. Closure properties of constraints. *Journal of the ACM* 44(4):527–548, 1997.
- [24] B. Larose and L. Zádori. Taylor terms, constraint satisfaction and the complexity of polynomial equations over finite algebras. *International Journal of Algebra and Computation* 16(3):563–581, 2006.
- [25] P. Broniek. Solving equations over small unary algebras. In *Computational Logic and Applications (CLA’05)*, DMTCS Proceedings AF, pages 49–60, 2006.
- [26] T. Feder and M. Y. Vardi. The computational structure of monotone monadic SNP and constraint satisfaction: a study through Datalog and group theory. *SIAM Journal on Computing* 28(1):57–104, 1998.
- [27] M. H. A. Newman. On theories with a combinatorial definition of “equivalence”. *Annals of Mathematics* 43(2):223–243, 1942.
- [28] D. Lehmann, A. Pnueli, and J. Stavi. Impartiality, justice and fairness: the ethics of concurrent termination. In *ICALP*, LNCS 115, pages 264–277, 1981.
- [29] N. Francez. *Fairness*. Springer, 1986.
- [30] E. W. Dijkstra. Self-stabilizing systems in spite of distributed control. *Communications of the ACM* 17(11):643–644, 1974.

- [31] E. W. Dijkstra. Self-stabilization in spite of distributed control. Manuscript EWD426, 1974. <https://www.cs.utexas.edu/~EWD/transcriptions/EWD04xx/EWD426.html>.
- [32] W. Fokkink, J.-H. Hoepman, and J. Pang. A note on K -state self-stabilization in a ring with $K = N$. *Nordic Journal of Computing* 12(1), 2005.
- [33] S. Qadeer and N. Shankar. Verifying a self-stabilizing mutual exclusion algorithm. In *Programming Concepts and Methods (PROCOMET)*, IFIP, pages 424–443, 1998.
- [34] K. Altisen, P. Corbineau, and S. Devismes. A framework for certified self-stabilization. *Logical Methods in Computer Science* 13(4), 2017.
- [35] V. B. F. Gomes, M. Kleppmann, D. P. Mulligan, and A. R. Beresford. Verifying strong eventual consistency in distributed systems. *Proceedings of the ACM on Programming Languages* 1(OOPSLA):109:1–109:28, 2017.
- [36] H. Ackermann, P. W. Goldberg, V. S. Mirrokni, H. Röglin, and B. Vöcking. Uncoordinated two-sided matching markets. *SIAM Journal on Computing* 40(1):92–106, 2011.
- [37] A. Tamura. Transformation from arbitrary matchings to stable matchings. *Journal of Combinatorial Theory, Series A* 62(2):310–323, 1993.
- [38] J. Chen, P. Skowron, and M. Sorge. Matchings under preferences: strength of stability and trade-offs. arXiv:1906.12274, 2019.
- [39] D. E. Knuth. *Mariages stables*. Les Presses de l’Université de Montréal, 1976.
- [40] A. E. Roth and J. H. Vande Vate. Random paths to stability in two-sided matching. *Econometrica* 58(6):1475–1480, 1990.
- [41] Apache CouchDB issue COUCHDB-988, 2010. <https://issues.apache.org/jira/browse/COUCHDB-988>.
- [42] Apache CouchDB issue COUCHDB-1649, 2013. <https://issues.apache.org/jira/browse/COUCHDB-1649>.
- [43] Apache CouchDB issue 6162: replication result depends on whether the target has a validate_doc_update function. <https://github.com/apache/couchdb/issues/6162>, 2026.
- [44] Apache CouchDB pull request 6168: fix revision patching for replicated changes. <https://github.com/apache/couchdb/pull/6168>, 2026.
- [45] D. Blackwell. gsm: governed state machines. <https://github.com/blackwell-systems/gsm>, version 0.16.0, 2026.
- [46] J. M. Hellerstein. The declarative imperative: experiences and conjectures in distributed logic. *SIGMOD Record* 39(1):5–19, 2010.
- [47] T. J. Ameloot, F. Neven, and J. Van den Bussche. Relational transducers for declarative networking. *Journal of the ACM* 60(2):15:1–15:38, 2013.
- [48] V. Balegas, S. Duarte, C. Ferreira, R. Rodrigues, and N. Preguiça. IPA: invariant-preserving applications for weakly consistent replicated databases. *Proceedings of the VLDB Endowment* 12(4):404–418, 2018.

- [49] P. E. O’Neil. The escrow transactional method. *ACM Transactions on Database Systems* 11(4):405–430, 1986.
- [50] D. B. Terry, M. M. Theimer, K. Petersen, A. J. Demers, M. J. Spreitzer, and C. H. Hauser. Managing update conflicts in Bayou, a weakly connected replicated storage system. In *SOSP*, pages 172–183, 1995.
- [51] P. Cousot and R. Cousot. Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *POPL*, pages 238–252, 1977.
- [52] F. Robert. *Discrete Iterations: A Metric Study*. Springer, 1986.

A Method

Names as statements. Every result cited in the paper is a **Theorem**, **Lemma** or **Corollary** in the development, cited by its Coq name. Each statement in the paper was read from the Coq source; a script checks that every name cited on the atlas and the website is declared and gated, and that the counts shown there equal the gate.

The gate. `coq/verify.sh` runs **Print Assumptions** on every gated result and fails if any depends on an axiom or an admitted proof, or if the number of gated results falls below the threshold. Nothing in the development is admitted.

The freeze. For the tests of Section 10, the eight kernel files were frozen: their SHA-256 hashes are recorded in `research/predictions/frozen.sha256`, and `docs/atlas/check.py`, run in continuous integration, fails if any of them changes. Outside recoveries and predictions are new files only.

Predictions. A prediction is committed as a file, unchanged after the commit that adds it, before any test; the commit is its timestamp. Candidates that were refuted before commitment are kept in `research/predictions/candidates.md`.

B Module index

File	Role
GovernanceConverse.v	the single-registry theorem (<code>cc_exact</code>), causal exactness
CanonicalExecution.v	state descent, the peak classification
AtomicRepair.v	repair run to completion after each event: <code>atomic_exact</code> , <code>atomic_derived_from</code> , <code>atomic_not_descent</code>
PresentedExecution.v	layer A: presented executions, the local and closure principles
CanonicalRecurrence.v	layer C: the fair-lasso kernel C1 and C2
TransportCSP.v,	layer B: sections as constraint solutions, polymorphisms, the chain and diamond cells
TransportCSPHard.v	
LayerInterfaces.v	the transfer lemma, the interfaces and the hierarchy across scales
NormalizationDerived.v	the single-registry theorem derived from the kernels
CRDTBoundary.v,	the compensation-free boundary
CvRDTEExact.v	
SelfStabilization.v,	outside theorems with the kernels frozen
InvariantConfluence.v,	
CRDTRecovery.v	
StableMatchingP2.v	prediction P2

The full index by regime is in `coq/README.md`.

C Recovery tables

Layer C bridges. Lines of proof for each recovered settlement result, and of its no-bad-lasso argument, over a shared toolkit of about 60 lines.

Recovered result	Whole bridge	No-bad-lasso argument
<code>chaotic_reaches_lfp_lasso</code>	36 lines	11 lines
<code>signed_settlement_lasso</code>	about 75 lines	5 and 6 lines
<code>signed_fidelity_lasso</code>	28 lines	5 lines
<code>fairflush_lasso</code> , <code>fairflushR_lasso</code>	3 and 4 lines	(the originals are C2)

Outside theorems. Approximate lines of Coq for each part, as recorded in the development.

Target	Parts
Self-stabilization	bridge from Theorem 3.8: 1 line; ring closure: 55; ring loop-freedom: 220; finite packaging: 100; adapter for systems that never quiesce: 110
Invariant confluence	polymorphism bridge: 44; exact form: 54; counterexample: 66; sufficiency: 117; replica model: 101
CRDT SEC	op-based bridge: 83 (kernel chain 36); Gomes et al.’s statement: 114; state-based kernel step: 21; semilattice facts: 24; merge reduction outside the kernels: 73

Layer A adequacy. For each delivery model of Table 1, the per-model cost is the proof that its notion of the same delivery equals the generated equivalence of its presentation; for at-least-once delivery it is about 250 lines.

D The regime map by family

Family	Regimes	Exact	Hard	Open	Excluded
Single registry	4	4			
Delivery	5	5			
CRDTs and streams	6	5			1
Acyclic federations	5	5			
Distributed propagation	6	5		1	
Channels and reconfiguration	8	6		2	
Monotone cycles	5	4			1
Invertible networks	5	3	1	1	
Lossy networks	8	2	3	3	
Composition	5	4		1	
Structure	3	2		1	
Total	60	45	4	9	2

The nine open tiles correspond to the seven open convergence questions of Section 11 and the open sheaf-assembly item. Each regime’s theorem is listed in `REGIME-AUDIT.md`.

E The committed predictions

The claims below are condensed from the committed files, `research/predictions/P2.md` and `P3.md`, with the Markdown removed.

P2 (committed 2026-10-11, commit ba043e9). *Model.* The marriage market with n men and n women and complete strict preferences; states are matchings (partial allowed); labels are the n^2 pairs; the update for (m, w) matches m with w if (m, w) blocks the matching, meaning each is single or prefers the other to the current partner, and their former partners become single; otherwise it changes nothing. A fair schedule names every pair infinitely often, naming a non-blocking pair being a no-op. *Claim.* There are $n \leq 4$, preferences with exactly one stable matching, and a start matching from which a bad fair lasso exists: a pair-word reaches a matching, and a nonempty pair-word that names every one of the n^2 pairs returns it to itself, with some loop state unstable. *Pass.* An explicit instance, its unique stable matching and a bad fair lasso, all mechanized. *Fail.* An exhaustive check, up to relabeling men, women and the sides, showing that for every $n \leq 4$ and every profile with exactly one stable matching there is no bad fair lasso from any start. *Probability stated:* about 0.6. *Outcome:* passed (Section 10.2).

P3 (committed 2026-10-11, commit aaba8b4). *Setting.* One document with revisions 1-a, 2-b, 3-c and 4-d, where 4-d is a deletion; each revision delivered once with its full history, one per request, to a fresh document id with `revs_limit` 3, in each of the 24 orders; CouchDB 3.5.3 and PouchDB 9.0.0. *Claims, each scored separately.* (1) CouchDB with default settings: the document ends live, with winning revision 1-a, on exactly the 6 orders in which 1-a is delivered last, and deleted on the other 18. (2) CouchDB with `stem_interactive_updates = false`, then compaction: deleted on all 24 orders. (3) PouchDB with default settings: live on exactly the 12 orders in which 1-a is delivered after 4-d, and deleted on the other 12. *Pass and fail.* Each order is run three times in a fresh database; orders whose three runs disagree are excluded and counted; a claim passes if every scored order matches the computed table and fails on any reproducible mismatch. *Probability stated:* about 0.75. *Outcome:* passed on all three claims; the packaging

deviation (a Homebrew build of the same version instead of the official image) is recorded in the outcome.