A Mechanized Metatheory for Mergeable Replicated Data Types:

RA-linearizability under three-way merge — eight verification conditions, a conditioned generalization, and the tombstone-free RGA end-to-end

Sal metatheory note
FP Launchpad, IIT Madras

July 7, 2026

Abstract

Mergeable replicated data types (MRDTs) resolve concurrent divergence by a *three-way* merge $\mathsf{mergeL}(l, a, b)$: the two divergent states a, b are reconciled relative to their lowest common ancestor l in a git-like version DAG. The Neem methodology reduces RA-linearizability of an MRDT to a fixed catalogue of per-data-type verification conditions (VCs), tied together by a once-and-for-all soundness meta-theorem. Mechanizing that meta-theorem in Lean 4, we find that its central induction is unsound as written — a reachable, fully LCA-legal execution of a legal add-wins MRDT defeats every bottom-up peel the proof can make — that the paper's own LCA lemma, while true, has a gap in its proof, and that no repair can lean on either of the two obvious crutches: the LCA argument cannot be forgotten (the counter MRDT provably admits no binary, CRDT-style collapse), and no lattice-flavoured contract is available (the lattice laws are equations over free tuples — the wrong instances for an LCA-aware merge — and no usable order survives). What works is a *delta* discipline: three unconditional relational VCs on rc, one merge symmetry, three *feasibility-conditioned* redistribution laws, and a single contextual *causal-delta* equation — eight VCs in all — from which every reachable configuration of the replicated store is RA-linearizable, *per version*. All of it is mechanized in Lean 4 with zero `sorry`s, and the VC set is validated in the strongest currency available: eleven of the twelve production MRDTs of the Sal repository — OR-sets, an enable-wins flag, counters, grow-only set and map, a tombstone RGA, Peritext, the multi-valued register and the add-wins priority queue — carry kernel-checked end-to-end RA-linearizability theorems through it. The one data type provably *outside* the VC set — the tombstone-free RGA, whose commutation holds only on reachable states — is hosted by a *conditioned* extension of the metatheory grown from this note's own open questions: a kernel-checked end-to-end theorem gives RA-linearizability *up to observational equivalence* at every reachable configuration, from a single honest-delivery assumption. This note is the complete, self-contained account: what an MRDT is and how to describe one (§2), the corrected flat metatheory and its validation (§3–§9), the conditioned metatheorem with its pen-and-paper end-to-end proof for the tombstone-free RGA (§10–§14), the open questions, and the mechanization map.

Contents

1 Introduction

An MRDT execution looks like a git history: replicas fork, apply operations locally, and merge. Unlike a state-based CRDT, whose binary merge must reconcile two states with no memory of their common past, an MRDT's merge receives a third argument — the state at the *lowest common ancestor* (LCA) of the two versions being merged — and may use it to compute *deltas*: what each side did since the fork. The paradigmatic example is the counter,

$$\mathsf{mergeL}(l, a, b) \;=\; a + b - l,$$

which adds the two sides' increments-since-l instead of double-counting their common past (Figure 1).

The Neem methodology (OOPSLA 2025) promises a clean division of labour: the data-type designer discharges a fixed catalogue of equations over do, mergeL and the conflict-resolution relation rc — mostly by SMT — and a soundness meta-theorem, proved once, converts those VCs into *RA-linearizability* of every reachable configuration. This note documents what a Lean 4 mechanization of that meta-theorem, in the full **ternary, version-DAG** setting, found: the meta-theorem's central induction is unsound as written, and for LCA-sensitive data types several of the catalogue's merge VCs are false in principle when quantified over all states. It also develops

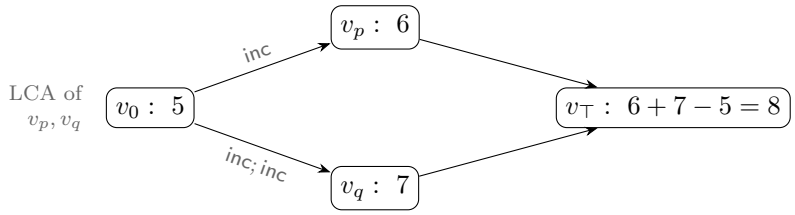

Figure 1: The counter MRDT. Both branches inherit the 5 from the fork point; the LCA argument lets the merge count each branch's *delta* exactly once. No binary merge of 6 and 7 alone can recover 8: the information "how much is shared" lives in the DAG, and the LCA argument is how the data type reads it.

the corrected metatheory: a set-relative linearization order, canonical states $\sigma(E)$ (each event set determines a unique state), and a ternary *Join Lemma* whose induction consumes a small, contextual VC surface. Its contributions:

1. **The paper's LCA lemma is true but its proof has a gap** (§4.1): $L(v_\ell) = L(v_1) \cap L(v_2)$ holds, but only as a *reachability invariant* of the store, whose proof needs a generator-uniqueness induction the paper silently assumes. We mechanize the transition system and the invariant.

2. **The soundness proof has a fully LCA-legal defeater** (§4.2): a reachable configuration of a legal add-wins MRDT — its final merge sitting at an honest LCA — on which every bottom-up peel of the paper's derivation rules is stuck. The meta-theorem must be rebuilt, not patched.

3. **The lattice contract does not transfer; a delta contract does** (§6): the lattice laws are equations over *free* tuples — the wrong instances for an LCA-aware merge. Side-idempotence $\mathsf{mergeL}(l, a, a) = a$ with l free fails for the counter $(2a - l)$, but its only DAG-honest instance has $l = a$ (the LCA of a version with itself *is* that version), where it holds trivially; and when two *distinct* histories reach equal side states, $2a - l$ is the *correct* merge — both branches' increments count — not an idempotence failure. The induced orders degenerate, and the honest associativity law has *four distinct LCAs* and holds only on feasible tuples. The moral: the algebra of an LCA-aware merge is an algebra of *causal histories*, not of concrete states. What replaces the lattice laws is a two-law *delta contract* — redistribution identities in which the LCA slot does, by arithmetic, the job idempotence does order-theoretically for CRDTs.

4. **Eight VCs suffice** (§7–§8): three relational VCs on rc (one of them carrying a different-replica guard — same-replica pairs are ordered by program order, and demanding rc cover them is unsatisfiable for real data types), merge symmetry, three feasibility-conditioned delta laws, and one contextual *causal-delta* equation. The adequacy theorem is per *version*: every version in the store, LCAs included, linearizes.

5. **The VC set is validated on the production catalogue** (§9): eleven of the twelve MRDTs shipped in Sal — OR-set, an optimized OR-set, enable-wins flag, grow-only set and map, increment-only and PN counters, a tombstone RGA, Peritext, the multi-valued register and the add-wins priority queue — are discharged end-to-end, kernel-checked; the

twelfth, the tombstone-free RGA, goes through the conditioned framework. The sweep produces an empirical *class map* of which data types need which strength of contract, with two surprises: tombstones are a metatheoretic trivializer, and commutativity class and delta class are independent axes.

6. **The eight VCs are one instance of a closure-indexed contract** (§15): a mechanized investigation of the note's own open questions establishes that the "second route" asymmetry (§8) is not a defect to be reunified but a *closure parameter* the data type declares; a single soundness bridge covers both strengths, and *all nine* production discharges route through it unchanged. The same investigation refutes the obvious attacks on the two remaining frontiers — it defeats even the commuting 2-OP rule on the defeater, pins the feasible-update-layer feasibility notion, and characterizes exactly why the tombstone-free RGA resists (it converges, but through semantic commutation the pointwise bubble cannot see). Pushed to its end, the investigation *hosts* the tombstone-free RGA: the swap architecture itself is refuted at merge unions, and a conditioned metatheory — an observational quotient, a canonical-witness discipline per version, and a generation discipline derived from born accuracy — proves it RA-linearizable up to observational equivalence, end-to-end under honest delivery. All kernel-checked.

Everything stated as a theorem below is mechanized with zero `sorry`s; §16 maps statements to Lean names and files (foundations under `Framework/`, metatheorems under `Metatheory/`, negative results under `Refutations/`, the per-RDT instances under `MRDT_Instances/`, the investigation record under `Development/`; each cited theorem kernel-checked).

2 Mergeable replicated data types, by definition and example

Data type and implementation. A replicated data type of *type* $\tau = \langle O_\tau, Q_\tau, \mathit{Val}_\tau \rangle$ exposes a set of update operations O_τ, a set of query operations Q_τ, and a domain of query return values Val_τ. An *MRDT implementation* of τ is a tuple

$$\mathcal{D} \;=\; \langle \Sigma,\ \sigma_0,\ \mathsf{do},\ \mathsf{mergeL},\ \mathsf{query},\ \mathsf{rc} \rangle,$$

where Σ is the space of replica states with initial state σ_0; $\mathsf{do} : \Sigma \times \mathcal{E} \to \Sigma$ applies an *event* — an update-operation instance $e = (t, r, o)$ carrying a globally unique timestamp t, its origin replica r, and the operation $o \in O_\tau$; $\mathsf{mergeL} : \Sigma^3 \to \Sigma$ is the three-way merge, whose first argument is the state at the lowest common ancestor of the two versions being merged; $\mathsf{query} : \Sigma \times Q_\tau \to \mathit{Val}_\tau$ reads a state; and $\mathsf{rc} \subseteq O_\tau \times O_\tau$ is the *conflict-resolution* relation, consulted to order concurrent non-commuting operations (it is the data-type designer's declaration of who wins a race). Queries never change state and play no role in the metatheory; the theory below reads the implementation as $\langle \Sigma, \sigma_0, \mathsf{do}, \mathsf{mergeL}, \mathsf{rc} \rangle$.

Describing an MRDT: the counter. To describe an MRDT one gives the five components and, when operations can race, says who wins. The increment-only counter of Figure 1 is the smallest complete example:

$$\Sigma = \mathbb{Z}, \qquad \sigma_0 = 0, \qquad \mathsf{do}(\sigma, (t, r, \mathsf{inc})) = \sigma + 1, \qquad \mathsf{mergeL}(l, a, b) = a + b - l, \qquad \mathsf{rc} = \emptyset.$$

All operations commute, so rc is empty — yet the merge genuinely uses its LCA argument: $a + b$ alone would double-count the history shared by the two branches, and the $-\,l$ subtracts it exactly once. This "read the shared past off the LCA" pattern recurs throughout the catalogue.

A data type with races: the OR-set. The add-wins observed-remove set stores tagged elements: add inserts a tag unique to the adding event, rem deletes exactly the tags it has *seen*. Sequentially, a remove after an add deletes it; concurrently the designer declares the add the winner: $\mathsf{rem} \xrightarrow{\mathsf{rc}} \mathsf{add}$. The merge keeps a tag iff both sides have it (nobody deleted it) or some side added it since the fork:

$$\mathsf{mergeL}(l, a, b) \;=\; (a \cap b) \cup (a \setminus l) \cup (b \setminus l).$$

The LCA here plays the role a tombstone set plays for the CRDT OR-set: deletion is absence *relative to* l, and the state needs no graveyard.

The generalized signature. Both examples' operations make sense on *every* state: any state can be incremented, any tag can be added. Such data types are *flat*, and for them the metatheory of §3–§9 quantifies its VCs over all states. But a data type whose operations carry *preconditions* — insert under a node that must exist, delete a node that must be live — breaks that quantification: its commutation equations are simply false at nonsense states. The full signature therefore extends the tuple with three declarations (`ConditionedMRDTSig`; formalized in §11):

$$\mathcal{D} \;=\; \langle \Sigma, \sigma_0, \mathsf{do}, \mathsf{mergeL}, \mathsf{rc}, \;\mathsf{Inv}, \;\mathsf{app}\rangle \quad \text{together with} \quad \approx \;\subseteq \Sigma \times \Sigma,$$

where Inv is a state invariant (a shape over-approximation of reachability), app an applicability predicate (when an event is sensible at a state), and $\approx$ an observational equivalence (what queries can distinguish; representation residue is quotiented away). Commutation is then required only at Inv-states where both events are applicable. Flat data types take $\mathsf{Inv} = \mathsf{app} = \top$ and $\approx \;=\; =$, collapsing the general signature to the plain one — one framework, of which §3–§9 is the flat instance and §10 onwards the general case.

The running hard example: the tombstone-free RGA. The Replicated Growable Array is the list CRDT/MRDT underlying collaborative text editing: characters are nodes identified by timestamps, each inserted *after* an anchor node. Production RGAs keep deleted nodes as *tombstones* because later concurrent inserts may still anchor on them. The *tombstone-free* RGA deletes physically, and it is describable in the general signature:

- Σ: finite forests — partial maps from node ids to (element, anchor id), the root id 0 never stored; σ_0 the empty forest.
- Operations: $\mathsf{Ins}(e, p, a)$ inserts a fresh node with element e under anchor a, and $\mathsf{Del}(p, x)$ deletes node x; both carry a *recorded path* p — the anchor's (resp. target's) ancestor chain as the issuing replica saw it. do resolves the first live entry of $a :: p$ and attaches there; a delete removes x physically and *rehomes* x's children to x's nearest live ancestor.
- $\mathsf{mergeL}(l, a, b)$: OR-set survivorship on node ids (present in both sides, or born since the fork on one), each survivor re-anchored by climbing its recorded chain to the nearest survivor.
- rc: empty in the directional sense (`Either`) — every pair commutes, but only *conditionally*.
- Inv: well-formed forest (every stored anchor live or root), id-monotone anchors ($\mathsf{anc}(x) < x$), root unstored.
- app: *accurate* (the recorded path is the target's true live ancestor chain in the current state) and *fresh* (the id is unused) — the predicate that holds at an op's generation but not at arbitrary replay states, which is where conditioning bites.
- $\approx$: equal live forests — same nodes, elements and anchors.

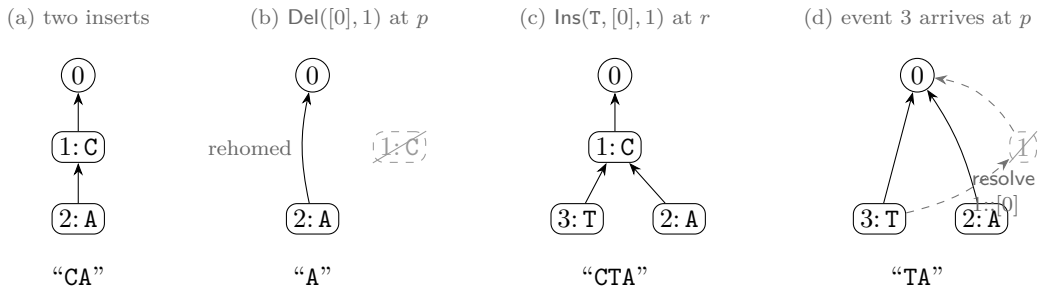

Figure 2: The running example as trees and the text they represent. Boxes are nodes *id*: *element*; arrows point to the anchor (anc); the document is the depth-first traversal visiting siblings newest-id first. (a) Two inserts build the chain $0 \leftarrow 1 \leftarrow 2$: text "CA". (b) Replica p deletes node 1 *physically* — no tombstone remains — and its child 2 is rehomed to 1's nearest live ancestor, the root. (c) Concurrently, replica r — which has not seen the delete — inserts T after C: event $(3, r, \mathsf{Ins}(\mathtt{T}, [0], 1))$, whose recorded path 1::[0] is 1's live ancestor chain at generation; at r the text reads "CTA". (d) When event 3 arrives at p the anchor is gone; resolve climbs the recorded path past the dead 1 to 0 and attaches there. Folding the two concurrent events in either order yields this same forest ("TA") — the pair commutes at accurate states, which is exactly what the recorded path buys and what no static rc policy could provide.

A three-node example (Figure 2) shows why every piece is needed. Start from the chain $0 \leftarrow 1 \leftarrow 2$ (node 1 under the root, node 2 under 1). $\mathsf{Del}([0], 1)$ removes node 1 and rehomes 2 to the root. A *concurrent* event $(3, r, \mathsf{Ins}(e, [0], 1))$ — generated before the delete was seen, anchored on 1, whose recorded chain $1 :: [0]$ is 1's live ancestor chain at generation — arrives after it: the anchor is gone, and the recorded chain is exactly what lets do resolve the insert to 1's parent 0, where the delete rehomed 1's children. Commutation of this pair over *all* states is false (the recorded path can lie about a nonsense state); at reachable, accurate states it holds. This data type is the reason the general signature exists, and §10–§14 prove it RA-linearizable up to $\approx$, end-to-end.

3 The ternary setting

This section fixes the model: the data-type signature, the replicated store and its transitions, and the specification the metatheorem must deliver. The definitions are the paper's, with one correction flagged where it occurs.

Signature. An MRDT is $\langle \Sigma, \sigma_0, \mathsf{do}, \mathsf{mergeL}, \mathsf{rc} \rangle$ as in §2, read flat for now ($\mathsf{Inv} = \mathsf{app} = \top$, $\approx \; = \; =$; the general case resumes in §10). We write $e(\sigma)$ for $\mathsf{do}(\sigma, e)$, and $e_1 \rightleftarrows e_2$ when $e_1(e_2(\sigma)) = e_2(e_1(\sigma))$ for every σ.

Executions. A configuration carries a *store*: a DAG of versions, each holding a state and the set of events that produced it, with per-replica heads. Transitions fork a replica, apply a fresh event at a head, query, or **merge**: pick heads v_1, v_2, find their LCA v_ℓ in the DAG, and install a new version with state $\mathsf{mergeL}(\sigma_{v_\ell}, \sigma_{v_1}, \sigma_{v_2})$ and event set $L(v_1) \cup L(v_2)$. Visibility $\xrightarrow{\mathsf{vis}}$ records what each event's issuing replica had seen.

RA-linearizability, per version. Write $e_1 \parallel e_2$ when neither $e_1 \xrightarrow{\mathsf{vis}} e_2$ nor $e_2 \xrightarrow{\mathsf{vis}} e_1$ (concurrency).

Definition 3.1 (linearization order, set-relative)**.** *For an event set E, the order lo^E puts e_1 before e_2 iff*

$$\left(e_1 \xrightarrow{\mathsf{vis}} e_2 \ \wedge\ e_1 \not\rightleftarrows e_2\right) \ \vee\ \left(e_1 \parallel e_2 \ \wedge\ e_1 \xrightarrow{\mathsf{rc}} e_2 \ \wedge\ \neg\, \exists e_3 \in E.\ e_2 \xrightarrow{\mathsf{vis}} e_3 \wedge e_2 \not\rightleftarrows e_3\right) :$$

a visible non-commuting pair is ordered by visibility; a concurrent pair whose conflict rc *resolves is ordered by* rc *— unless e_2 is already absorbed (overwritten) by a later non-commuting event of E. The paper's order* lo *is the variant whose absorber clause ranges over the whole configuration; since a* lo*-edge asserts no absorber anywhere, $\mathsf{lo} \subseteq \mathsf{lo}^E$ pointwise, for every E.*

Definition 3.2 (RA-linearizability)**.** *A list $\pi = e^1 e^2 \cdots e^n$ witnesses a version holding state s and event set E when (i) π enumerates E without repetition; (ii) π extends* lo*: $i < j$ implies $\neg(e^j\ \mathsf{lo}\ e^i)$; and (iii) π folds to the state: $e^n(\cdots e^2(e^1(\sigma_0))\cdots) = s$. A configuration is RA-linearizable when every version in the store — not only the replica heads; LCAs are historical versions and the merge reads them — admits a witness. In the general signature the fold clause (iii) is read up to the data type's observational equivalence, $e^n(\cdots e^1(\sigma_0)\cdots) \approx s$ — the same definition with its equality parameter generalized; flat data types instantiate $\approx\ =\ =$ and this section's form is exactly that instance.* `IsRALinearizable3, IsRALinearizable3Eq`

Why a set-relative order? One definitional correction is forced at the outset: for every internal use, the absorber clause must range over the event set of *the version being linearized*, not over the whole configuration — an event can gain absorbers from another branch, silently cancelling an order edge inside a version whose own state depends on it (the defeater of §4.2 realizes exactly this). We therefore construct witnesses respecting lo^E throughout; since $\mathsf{lo} \subseteq \mathsf{lo}^E$, every such witness also extends the paper's order, and Definition 3.2 is delivered exactly as the paper states it.

Running examples. Three data types exercise every corner of the theory:

- the **counter**: $\Sigma = \mathbb{Z}$, $\mathsf{inc}(\sigma) = \sigma + 1$, $\mathsf{mergeL}(l, a, b) = a + b - l$; all operations commute, but the merge genuinely uses its LCA;
- the **OR-set** (tagged add-wins set): elements are tagged by the adding event's timestamp; rem deletes the tags it has seen; concurrently, $\mathsf{rem} \xrightarrow{\mathsf{rc}} \mathsf{add}$ resolves in favour of the add. The merge keeps an element iff *both* sides have it or *some* side added it since the fork:

  $$\mathsf{mergeL}(l, a, b) \ =\ (a \cap b) \cup (a \setminus l) \cup (b \setminus l)$$

  — pointwise, "if the tag is in l, require it in both branches (nobody deleted it); otherwise one branch suffices (somebody added it)." Here the LCA acts as a *tombstone set*: deletion is represented by *absence relative to l*, so the state itself needs no graveyard;
- the **enable-wins flag**: per-replica pairs (counter, flag); enabling bumps your own counter and sets your flag; disabling clears all flags; the merge compares each replica's counter against the LCA's to decide whether an enable "happened since the fork" on that replica. Its merge *compares* counters rather than accumulating sets — and that difference will be visible in the metatheory (§9).

4 Three facts the metatheory must respect

Before building anything, we record two boundary facts about the paper's development: its store-level LCA lemma is true but under-proved, and its soundness induction is refuted by a fully legal execution. Together they fix the shape of the corrected metatheory — resting on a mechanized store invariant (§4.1) and built around the join rather than the sides (§4.2).

4.1 The LCA lemma is a reachability invariant — with a gap in the published proof

Everything above silently used:

Lemma 4.1 (LCA lemma)**.** *In every reachable configuration, if v_ℓ is an LCA of versions v_1, v_2 in the store's DAG, then $L(v_\ell) = L(v_1) \cap L(v_2)$.*

This is what lets merge VCs be stated over *event sets*: the state the merge receives as l is the canonical state of exactly the intersection of the two sides' histories. The lemma is true — but it is a *global induction over the reachable store*, not a local fact. Its mechanized proof threads a store invariant (all parents allocated; event sets monotone along edges; and an *origin* invariant — each event appears first at a unique generator version) through every transition; the published proof asserts the generator-uniqueness step without the induction that sustains it. An erratum-sized gap: the statement survives, the proof as written does not. (A related boundary fact: *criss-cross* merges can defeat the *existence* of an LCA — they gate merge-enabledness, not the lemma. Open Question 15.6 takes up the extension.)

4.2 A fully LCA-legal execution defeats the soundness proof

The generic half of the paper's contract is a *bottom-up linearization* theorem: given linearization witnesses for the two sides of a merge, construct one for the merged version by repeatedly *peeling* a final event through mergeL using the catalogue's interchange VCs. One might hope the version DAG's discipline — every merge sitting at an honest LCA — keeps the peel supplied with peelable events. It does not: one staging replica suffices to realize the intersection of two histories as an honest version, and the merge induction is stuck.

Example 4.2 (the defeater)**.** *The data type is a one-key add-wins skeleton with tagged adds: $\sigma = (A, D) \in 2^{\mathbb{T}} \times 2^{\mathbb{T}}$ with live set $A \setminus D$; add_t inserts its tag into A; rem kills everything it currently sees ($D := A \cup D$ — the state-dependence is what makes $\mathsf{add} \not\rightleftharpoons \mathsf{rem}$); concurrently $\mathsf{rem} \xrightarrow{\mathsf{rc}} \mathsf{add}$ (add wins); the merge is componentwise union, ignoring its LCA argument. A legal MRDT. Replica p adds (A_p), replica q adds (A_q). A staging replica s merges both one-add versions, creating v_s with $L(v_s) = \{A_p, A_q\}$. Now p removes (R_p, seeing only A_p) and then merges with v_s; symmetrically q removes (R_q) and merges with v_s. Figure 3 shows the DAG. The final merge of the two heads finds* $\mathrm{LCA} = v_s$*, and $L(v_s) = \{A_p, A_q\} = L(\mathit{head}_p) \cap L(\mathit{head}_q)$ — the configuration is fully legal. Now count last events. p's head lives on exactly A_q's tag, so a state-correct witness must run R_p before A_q, and neither R_p nor A_p can come last: every linearization of p's head ends in A_q. Symmetrically, every linearization of q's head ends in A_p. Yet in the merged version every tag is dead, so its linearizations must end in one of the removes. The events the induction must peel are buried in the very sides that own them; no ordering of the paper's bottom-up rules applies.*

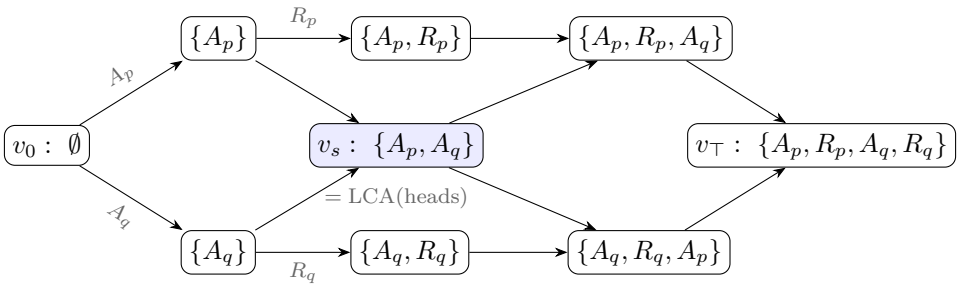

Figure 3: The defeater (Example 4.2). Nodes are versions labelled by their event sets; the shaded version v_s, created by a staging replica, realizes $L(\text{head}_p) \cap L(\text{head}_q)$ as an honest store version, so the final merge is LCA-legal. The removes R_p, R_q — the only events a bottom-up derivation may peel last — are buried mid-history on their own sides.

The two removes commute ($\mathsf{rem} \not\rightleftarrows \mathsf{add}$ but $\mathsf{rem} \rightleftarrows \mathsf{rem}$), which invites a particular rescue attempt: the paper's BottomUp-2-OP rule (`inter_lca_2op`) admits the commuting case and could, one hopes, peel a remove last. It cannot, for a reason that is a one-line state fact. To peel o last from a side, that side's state must be an o-image — of the form $\mathsf{do}(\sigma', o)$ with o outermost. But in the add-wins skeleton *every* rem-output has empty live set ($\mathsf{rem}(A, D) = (A, A \cup D)$), whereas both heads carry a live tag (head_p lives on A_q, head_q on A_p — the opposite add, merged in *after* the local remove). So neither head is a rem-image, and no bottom-up rule can extract a remove from it. The tempting linearization $[A_p, R_p, A_q, R_q]$ is a valid witness of the *merged* version, but its restriction to head_q folds to the all-dead state, not head_q's actual state (live A_p): it is not assembled from a state-correct side witness, which is exactly what the induction requires. All three facts are mechanized (`InterLca2op_Defeater_Arbiter`: `awset_rem_output_empty`, `no_inter_lca_2op_rem_peel_of_defeater`, `crack1_witness`).

The consequence for the design: the metatheorem cannot be patched peel-by-peel — it must be rebuilt on canonical states and a Join Lemma, with merge-side VCs that are statements *about the join*, not about how either side's state factors.

5 Canonical states and the ternary Join Lemma

We now rebuild — new proof architecture, and with it a *new set of verification conditions* to replace the paper's catalogue. The lesson of the defeater is architectural: witnesses for a merged version cannot be manufactured out of witnesses for its sides, so the corrected metatheory must not traffic in witness lists at all. The rebuild has two layers. The *update layer* makes states, not sequences, the objects of the induction: it assigns to every event set E a single well-defined state, its *canonical state* (Definition 5.2 below). The *merge layer* is then one equation between canonical states — the ternary *Join Lemma* — and establishing it from per-data-type VCs becomes the entire burden of the metatheorem (§6–§7).

The update layer comes first, and a structural observation makes it cheap: this half of the corrected theory never mentions the merge. Its one theorem is stated over the update signature $\langle \Sigma, \sigma_0, \mathsf{do}, \mathsf{rc} \rangle$ alone:

Theorem 5.1 (convergence)**.** *Assume the update-layer VCs (items 1–3 of §7). Then any two lo^E-respecting enumerations of a finite event set E fold from σ_0 to the same state.*

This is where the set-relative reading of §3 earns its keep: with the paper's configuration-wide order in place of lo^E, the theorem is *false* — the defeater's heads are already counterexamples, since two order-respecting replays of the three events of head_p reach different states. Convergence licenses a definition:

Definition 5.2 (canonical state)**.** *For a finite event set E, the canonical state $\sigma(E)$ is the state obtained by folding any lo^E-respecting enumeration of E from σ_0. At least one respecting enumeration exists because lo^E is acyclic (VC 2); the result is independent of the choice by Theorem 5.1; and $\sigma(\emptyset) = \sigma_0$. We write $\sigma(E)$ for the canonical state of an event set, and σ_v for the state a version v happens to hold in the store — proving that the two coincide is precisely the metatheorem's job.*

With canonical states in hand, witness lists disappear as promised: a version linearizes iff $\sigma_v = \sigma(L(v))$ — the witness, when one is owed, is any respecting enumeration of $L(v)$. The induction can therefore maintain a state-level invariant ("every version holds σ of its events") instead of constructing sequences. The one genuinely ternary statement is what the merge must produce:

Definition 5.3 (ternary Join Lemma)**.** *For backward-closed event sets E_1, E_2 of a reachable configuration,*

$$\sigma(E_1 \cup E_2) \;=\; \mathsf{mergeL}\big(\sigma(E_1 \cap E_2),\ \sigma(E_1),\ \sigma(E_2)\big).$$

By Lemma 4.1 the first argument is exactly the state the store hands the merge. The adequacy proof (§8) is then an induction over the transition system maintaining "every version holds the canonical state of its event set"; the merge case *is* the Join Lemma. What remains is the question this note exists to answer: which per-data-type VCs make the Join Lemma true?

6 Why the contract is a delta contract

Where should the laws that prove the Join Lemma come from? The obvious supplier is the CRDT tradition. For state-based CRDTs — binary merge, no LCA — merge-coherence is lattice-flavoured: associativity, idempotence, inflation of updates. None of the three transfers to the ternary merge: their free-tuple forms are the wrong equations for an LCA-aware merge, and their honest instances are too weak to power a lattice-style proof. This section works through why, because the shape of what fails to transfer reveals the contract that succeeds.

Idempotence: the free-tuple form is the wrong equation. The *diagonal* law $\mathsf{mergeL}(a, a, a) = a$ — the paper's own MergeIdempotence — holds (for the counter: $a + a - a = a$), but it is not the law a lattice-style proof consumes. The binary architecture uses idempotence to absorb a duplicated *side*, $\mathsf{merge}(a, a) = a$, with nothing tying the two copies to a common past; the ternary analogue would be $\mathsf{mergeL}(l, a, a) = a$ with l *free*. But quantifying l freely asks the merge about tuples the version DAG never produces. Merging a version with *itself* has $l = a$ — the LCA of a and a is a — where the law collapses to the diagonal and holds. And when two *distinct* histories happen to reach equal side states, feasibility puts l strictly below them and the counter's $\mathsf{mergeL}(l, a, a) = 2a - l$ is the *correct* answer: both branches' increments count. So there is no idempotence *failure* — there is no meaningful side-idempotence *law*: its honest instances are the trivial diagonal, and demanding it unconditionally would exclude precisely the LCA-sensitive data types the ternary theorem exists for.

The general principle, which the rest of this section instantiates: the algebraic properties of an LCA-aware merge are properties of the *causal history*, not of the concrete state. Two sides merge idempotently when their *histories* coincide — which is what forces $l = a$ — not when their states happen to be equal; equal states with different histories rightly merge differently. The lattice laws go wrong exactly by quantifying over states while forgetting the histories that produced them, and every law that survives below is stated over history-indexed data: canonical states $\sigma(E)$, feasible tuples (the LCA slot carrying the intersection history), downset deltas.

No usable order. The lattice-style workhorse order $x \sqsubseteq y \coloneqq \mathsf{merge}(x, y) = y$ has ternary candidates $x \sqsubseteq_l y \coloneqq \mathsf{mergeL}(l, x, y) = y$, but for the counter this relation degenerates to $x = l$; no antisymmetry survives to power order-theoretic reasoning. The single contextual VC below is accordingly an *equation*, not an inequality.

Associativity is the wrong target. The natural guess — $\mathsf{mergeL}(l, \mathsf{mergeL}(l, a, b), c) = \mathsf{mergeL}(l, a, \mathsf{mergeL}(l, b, c))$ — assumes one LCA for all three merges. In an execution, re-associating a three-way reconciliation involves *four* distinct LCAs:

$$\mathsf{mergeL}\big(l_{(ab)c},\, \mathsf{mergeL}(l_{ab}, a, b),\, c\big) \stackrel{?}{=} \mathsf{mergeL}\big(l_{a(bc)},\, a,\, \mathsf{mergeL}(l_{bc}, b, c)\big),$$

with $l_{ab} = \sigma(E_a \cap E_b)$, $l_{(ab)c} = \sigma((E_a \cup E_b) \cap E_c)$, and so on. For the counter — whose canonical states are additive measures over event sets — the two sides come to $a + b + c$ minus the two LCA measures, and they agree because

$$(E_a \cup E_b) \cap E_c = (E_a \cap E_c) \cup (E_b \cap E_c) \quad \Longrightarrow \quad \text{both LCA-sums} = |E_{ab}| + |E_{ac}| + |E_{bc}| - |E_{abc}|$$

by inclusion–exclusion. The law holds, but only *through the set-algebra of intersections*: over four unconstrained states it is false. Honest associativity is inherently a feasible-tuple fact — and, the mechanization's pleasant surprise, **the corrected induction never needs it**. Every merge the induction forms already sits at its honest LCA; the only laws consumed are two *redistribution* identities that equate two honestly-formed merges:

Definition 6.1 (the delta contract)**.**

$$\begin{aligned} \textit{LOCAL-REDISTRIBUTE}: \quad & \mathsf{mergeL}(l,\, \mathsf{mergeL}(m, x, c),\, y) \;=\; \mathsf{mergeL}(m,\, \mathsf{mergeL}(l, x, y),\, c), \\ \textit{REDISTRIBUTE}: \quad & \mathsf{mergeL}\big(\mathsf{mergeL}(m, x_0, c),\, \mathsf{mergeL}(m, x_1, c),\, \mathsf{mergeL}(m, x_2, c)\big) \\ & \qquad = \; \mathsf{mergeL}\big(m,\, \mathsf{mergeL}(x_0, x_1, x_2),\, c\big). \end{aligned}$$

Read c as a *delta relative to* m (in the induction, c will be "apply event e to its causal past"). LOCAL-REDISTRIBUTE says a delta application commutes past an enclosing merge through one branch slot. REDISTRIBUTE says a delta carried by *all three* components extracts *once* — the LCA slot cancels the duplicates by arithmetic. This is the delta contract's quiet coup: cancelling the duplicated shared contribution is exactly the job idempotence performs in lattice-based convergence arguments, now done without any idempotence — which is why the counter sails through (REDISTRIBUTE for the counter is the identity $\sum x_i + 3c - 3m - (x_0 + c - m) - \cdots$ collapsing on both sides).

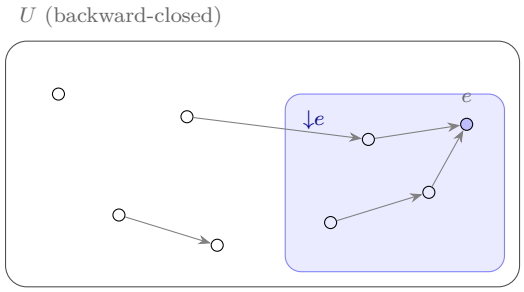

Figure 4: The causal-delta equation. The shaded region is e's causal past $\downarrow e$ (what e saw); e is lo^U-maximal. e's effect on the whole world A equals: compute e's effect on its own causal past B, then merge that delta into A *over LCA* B. The merge sits at its honest LCA: $(U \setminus \{e\}) \cap \downarrow e = \downarrow e \setminus \{e\}$. Concurrent context that e never observed cannot amplify what e does.

Where the contract holds unconditionally — and where it cannot. The two laws hold *on all states* exactly for the group-like class (the counter) and the lattice-like, LCA-blind class (grow-only structures). They are *unconditionally false* for the OR-set: take an element tag present in c and l but in neither branch — the two sides of LOCAL-REDISTRIBUTE then disagree about whether the LCA's copy survives. But such a tuple can never arise: a tag in a canonical LCA state is in *both* branches unless removed, and a removal is itself an event of the branch. The failing tuples are *infeasible*, and this is a theorem-grade phenomenon, not an inconvenience:

Finding. *For LCA-sensitive MRDTs beyond the group $\oplus$ lattice classes, the merge-coherence laws are irreducibly feasibility-conditioned: true on canonical tuples at honest LCAs, false in general. Quantifying merge VCs over all states — the published catalogue's style — is not merely inconvenient for such data types; for several laws it is wrong in principle. (The enable-wins flag even falsifies the unit law* $\mathsf{mergeL}(\sigma_0, \sigma_0, s) = s$ *on an infeasible state with a set flag and a zero counter.)*

Accordingly the delta laws enter the final contract in feasibility-conditioned form: quantified over canonical states of backward-closed event sets, with every mergeL node at its honest LCA. The unconditional form remains available as a one-line *on-ramp* (it trivially implies the conditioned form) for the group and lattice classes.

7 The eight verification conditions

The update-side obligations of §5 and the merge-side laws of §6 now assemble into the full contract — a *new* VC set, replacing the paper's catalogue of twenty-four. For an MRDT $\langle \Sigma, \sigma_0, \mathsf{do}, \mathsf{mergeL}, \mathsf{rc} \rangle$, the contract is:

Update layer (unconditional, SMT-shaped).

1. **rc-non-comm (guarded)** — for distinct events *from different replicas*: they fail to commute iff rc orders them in exactly one direction. *The conflict table is the commutativity table, with a direction.* The different-replica guard is a semantic necessity, not a convenience:

same-replica events are totally ordered by program order, so they are never concurrent and there is no conflict for rc to resolve. An unguarded VC would instead demand that rc order *every* non-commuting pair — unsatisfiable for real data types: the *optimized OR-set*'s same-replica same-element adds do not commute (the newer add evicts the older tag), yet they can never race, and its rc rightly ignores them. (The paper's F* artifact carries exactly this guard in its interface — `App_mrdt.fsti`: `requires distinct_ops o1 o2 /\ get_rid o1 <> get_rid o2`.)

2. **no-rc-chain** — conflict priorities do not chain: never $o_1 \xrightarrow{\mathsf{rc}} o_2$ and $o_2 \xrightarrow{\mathsf{rc}} o_3$. This is what makes the linearization order acyclic, so maximal events exist — the entire peel-and-recurse architecture stands on it.

3. **cond-comm** — a resolved conflict is invisible once an absorber runs: swapping an rc-ordered concurrent pair deep in a fold changes nothing for any later non-commuting event, across arbitrary intervening events. This is the engine of convergence.

Merge layer.

4. **merge symmetry** — $\mathsf{mergeL}(l, a, b) = \mathsf{mergeL}(l, b, a)$.

5. **feasible unit** — $\mathsf{mergeL}(\sigma_0, \sigma_0, \sigma(E)) = \sigma(E)$: merging over nothing is a no-op, on states that can arise.

6. **feasible local-redistribute** and

7. **feasible redistribute** — Definition 6.1, quantified over canonical tuples at honest LCAs.

8. **the causal-delta equation** (CDVC3) — for a lo^U-maximal event e of a backward-closed U, with $A = \sigma(U \setminus \{e\})$ and $B = \sigma(\downarrow e \setminus \{e\})$ the canonical state of e's own causal past:

$$\mathsf{mergeL}\big(B,\ A,\ e(B)\big)\ =\ e(A).$$

What an event does is determined by what it saw (Figure 4): applying e to the whole world equals computing e's delta on its causal past and merging it in over that past.

Items 1–4 are one-liners for every data type we have discharged; items 6–7 frequently hold unconditionally anyway (for the OR-set, REDISTRIBUTE is a pointwise Boolean tautology); essentially the entire per-data-type proof effort concentrates in item 8, whose discharge follows a uniform recipe: characterize $\sigma(E)$ (a "sandwich" invariant along canonical enumerations), then a *trichotomy* placing every event the maximal e interacts with either inside $\downarrow e$ or absorbed on a side that owns it.

Compare the published catalogue: 24 VCs per data type, plus five more the soundness proof uses silently. Seventeen of the 24 — the BOTTOMUP induction-scheme families — exist to drive the broken induction and are consumed nowhere in the corrected one; the paper's MERGEIDEMPOTENCE is likewise never used. The seventeen were, however, doing honest work of a different kind: they are an *automation layer*, Skolemizing "holds on feasible states" into base-and-step equations an SMT solver can discharge. Rebuilding that layer, aimed at the corrected target (CDVC3 rather than BOTTOMUP), is in our view the most valuable open engineering problem this work leaves (§15).

8 Adequacy

The contract delivers. Everything a data type owes is the eight equations of §7; everything else is proved once, for all data types, as part of the execution model.

Theorem 8.1 (soundness of the eight VCs)**.** *If an MRDT satisfies VCs 1–8, then every configuration reachable from the initial configuration in the ternary transition system is RA-linearizable, per version.* `ra_linearizable_of_core_feasible_cd3`

The proof shape: the update layer powers the set-relative convergence theorem, hence canonical states; the transition induction maintains "every store version is canonical for its event set" (fork and query are trivial; apply is the extend lemma); the merge case reduces, via Lemma 4.1, to the ternary Join Lemma, which is proved from VCs 4–8 by strong induction on $|E_1 \cup E_2|$: select a $\mathsf{lo}^{E_1 \cup E_2}$-maximal e, decompose each side containing e through $\downarrow e$ (LOCAL-REDISTRIBUTE at the honest LCA $\downarrow e \setminus \{e\}$), cancel the shared delta (REDISTRIBUTE), close the principal step with CDVC3, and recurse. The buried-event pathology of Example 4.2 dissolves structurally: no step ever asks how a *side's* state factors — every factoring happens through the join.

Everything else one might fear owing is proved once, for all data types, as an execution-model invariant: visibility transitivity, per-version backward closure, store well-formedness, and Lemma 4.1 itself. The data type owes exactly the eight equations.

A second route, and an honest asymmetry. One discharged data type does not fit the pattern above. The enable-wins flag's merge *compares counters* against the LCA, and for such merges the Join Lemma's hypotheses as stated (backward closure under non-commuting-visibility) are provably too weak: there is a closure-legal tuple — a live LCA enable, a dead post-LCA enable inflating one side's counter, the killing disable visible only to the other side — on which the production merge computes the wrong flag. Full *causal* closure (which the store invariant actually supplies) excludes it, and the flag is discharged through a full-closure variant of the join. The finding: **required closure strength is a function of the merge's shape** — commutation-closure suffices for set-shaped merges, counter-comparison merges need full causal closure. Reunifying the two routes was expected to be routine — "full closure only strengthens what a data type may assume." Mechanization shows it is not (§15, Open Question 15.4): the naive reunification — re-run the Join Lemma induction with full-closure hypotheses — is refuted, because no single-event (or block) peel preserves full closure. What survives is a *closure-indexed* contract in which the data type *declares* its closure strength; the soundness bridge is uniform across strengths, so no reunification into one strength is needed.

9 The catalogue, discharged

Table 1 is the empirical heart of the note: the VC set is not merely adequate in principle — the data types people actually ship satisfy it. Three lessons from the sweep:

Tombstones are a metatheoretic trivializer. The catalogue's celebrated hard cases — the RGA sequence type and the Peritext rich-text type — land in the *easiest* tier. The reason is almost embarrassing: with tombstones, deletion *adds* a record; every state component is grow-only; all operations commute; the merge is a blind join. All of the RGA's genuine difficulty lives in the *read-side* traversal (turning the node soup into a sequence), which RA-linearizability of the

MRDT	contract tier	end-to-end theorem	σ-characterization (one line)
Grow-only set	unconditional	`GOSet_ra_linearizable3_eq`	union of adds
Grow-only map	unconditional	`GOMap_ra_linearizable3_eq`	union of (k, v) records
Incr.-only counter	unconditional	`IOC_ra_linearizable3_eq`	#events
PN-counter	unconditional	`PN_ra_linearizable3_eq`	#inc − #dec
RGA (tombstone)	unconditional	`RGAM_ra_linearizable3_eq`	grow-only nodes + tombstones
Peritext	unconditional	`Peritext_ra_linearizable3_eq`	grow-only mark records
OR-set	feasible	`ORSet_ra_linearizable3_eq`	tag live iff added, no vis-later `rem`
OR-set (efficient)	feasible	`ORSetE_ra_linearizable3_eq`	… or same-replica `add` later (eviction)
Enable-wins flag	full-closure	`EWFlag_ra_linearizable3_eq`	per key: ctr = #enables; flag iff unkilled enable
Multi-valued register	feasible	`MVR_ra_linearizable3_eq`	tagged writes × overwrite log; all-comm ⇒ $B{=}\sigma_0$
Add-wins priority queue	feasible	`AWPQ_ra_linearizable3_eq`	the OR-set pattern on A; grow-only I
Bounded counter (escrow)	all-comm	`BC_ra_linearizable3_eq`	convergence flat; the *bound* is a conditioned safety theorem (`bc_version_inv`)
FWW reservation register	all-comm	`FWW_ra_linearizable3_eq`	payload arbitration (min-ts semilattice); winner characterized at every version (`fww_version_min`)
LWW register	all-comm	`LWW_ra_linearizable3_eq`	payload arbitration (max-ts semilattice), the last-writer twin of FWW; winner characterized (`lww_version_max`); *end-to-end mechanized*, unlike the flat CRDT LWW (sorry-carrying VC bridge)
BudgetCart	or-set $\supseteq_\tau$	`BCart_ra_linearizable3_eq`	budgeted cart, spend derived from live instances; safety *gated* on causal canonicity (OQ 15.8)
Mergeable queue (Peepul)	*direct join*	`queue_ra_linearizable3`	no $\supseteq_\tau$ exists (enqueue clique); the merge itself is the linearization witness
RGA (tombstone-free)	*conditioned*	`rga_tombstone_free_ra_linearizable3_eq` — RA-lin up to $\approx$, honest delivery (§15)	
Peritext (tombstone-free)	*composed*	`peritextComposed_ra_linearizable_up_to_eq` — RGA ⊗ marks through the product kit; 1,064 lines total	
Peritext (tombstone-free, fused)	*fused*	`peritext_ra_linearizable_up_to_eq` — one RGA at $\alpha :=$ `char` ⊕ `boundary`; convergence by instantiation, genuine positional intent (`render_id_active_iff_between`); 773 lines	

Table 1: The Sal production catalogue. Every instance concludes through the ONE generic conditioned metatheorem (§10–§14): the nine flat data types at the identity instantiation ($\approx = =$, their VC bundles feeding the flat collapse `FlatGeneric_Bridge.lean`, instance theorems in `MRDT_Instances/ (per-RDT)`), the tombstone-free RGA at full generality. The contract-tier column is each data type's Join-Lemma discharge content (`MRDT_Instances/`, one directory per RDT); the historical flat corollaries over the raw system remain there as internal steps.

state never touches. By contrast the *tombstone-free* RGA — which physically deletes and repairs positions by climbing ancestor paths — is the one data type the eight VCs cannot host, for a reason worth stating precisely: its commutation VC is itself *reachability-conditioned* (two inserts commute only on well-formed states), and the update layer above, like the paper's, quantifies over all states. That was the question the tombstone-free RGA was built to force, and it is now answered — not by a conditioned update *layer* (every layer-level conditioning is refuted, see the feasible-update question in §15) but by a conditioned *metatheory* that abandons pointwise swaps altogether; the resulting end-to-end theorem is RA-linearizability up to observational equivalence under honest delivery.

Commutation class and delta class are independent axes. The multi-valued register is all-commuting, yet *not* in the unconditional tier: its merge is intersection-shaped, $(l \cap a \cap b) \cup (a \setminus l) \cup (b \setminus l)$, and fails the delta laws on infeasible tuples exactly as the OR-set does. The boundary between tiers is drawn by *how the merge uses its LCA*, not by whether operations commute. (The compensation: all-commuting makes every punctured causal past empty, $B = \sigma_0$, so the feasible discharge needs no trichotomy at all.)

The eight VCs certify real code on its historical fault line. The enable-wins flag's repository history contains a known-broken sibling: a global-counter variant whose compositional VC (`inter_right_1op`) fails on DAG-shaped executions, the very defect per-replica counters were introduced to fix. The mechanized σ-facts prove the production merge computes exactly union-liveness — *verified on precisely the corner where the sibling fails.* This is the methodology working as intended: the metatheory does not merely bless the catalogue, it explains *why* the fixed design is right.

10 The conditioned metatheorem

The eight VCs of §7 quantify over *all* states, and §9 showed that is right for flat data types and fatal for state-dependent ones: in the tombstone-free RGA of §2, two concurrent inserts commute only when each one's anchor is faithfully resolvable in the other's presence, and an insert under a since-deleted, physically-removed node is not even applicable. Written over all states, the

commutation VC is simply false (a mechanized counterexample: `naive_general_swap_false`). The remedy is the general signature of §2: condition every VC on the state invariant Inv and the applicability predicate app, relax the witness fold of Definition 3.2 to the data type's observational equivalence $\approx$, and prove RA-linearizability from the conditioned VCs once and for all. The developer's burden is only to describe sensible operations; the metatheorem does the rest. This section states the conditioned VCs and the metatheorem with its pen-and-paper proof; §13 discharges the VCs for the tombstone-free RGA; §14 records how the mechanization actually went — including the machine-checked refutations that reshaped the route — and what the completed assembly proves.

11 The conditioned verification conditions

A datatype presents a signature $\langle \mathrm{State}, \sigma_0, \mathsf{do}, \mathsf{merge}, \mathsf{Inv}, \mathsf{app}\rangle$ where Inv is a predicate on states and app a predicate on (operation, state) pairs (Lean: `ConditionedMRDTSig`, `Framework/MRDTSig.lean`). It must supply:

VC 1 (Discipline)**.** $\mathsf{Inv}\,\sigma_0$ *holds;* do *preserves* Inv *on applicable operations; and generation is disciplined: every operation is* app *and fresh at its birth state, with globally unique, monotone ids. (This is the execution model, Lean:* `ConditionedConfiguration`*,* `Framework/ConditionedExecutionModel.lea` *— kernel-clean, axiom-free.)*

VC 2 (Conditioned commutation — the swap witness)**.** *For concurrent operations* a, b *and any state* σ *with* $\mathsf{Inv}\,\sigma$ *at which both are applicable in the relevant reorder sense,* $\mathsf{do}\,(\mathsf{do}\,\sigma\,a)\,b \approx \mathsf{do}\,(\mathsf{do}\,\sigma\,b)\,a$.

VC 3 (Threadable reorder invariant)**.** *An auxiliary invariant* J *on (state, pending-event) that (i) certifies applicability/the swap precondition of VC 2 at reorder states, (ii) holds at the enabling fold of each event, and (iii) is preserved by each reorder step. Only the enabled events — those whose causal past is folded — need be certified; that scope is exactly what a linearization repair ever transposes.*

VC 4 (Merge–linearization bridge — the conditioned Join Lemma)**.** *For a reachable LCA state* l *and branches* a, b *obtained from* l *by disjoint concurrent event lists,* $\mathsf{merge}\,l\,a\,b \approx \mathsf{fold}\,l\,\pi$ *for a* lo^E*-respecting enumeration* π *of the branch events.*

For flat datatypes $\mathsf{Inv} = \mathsf{app} = \mathsf{true}$, VC 2 degenerates to unconditioned commutation, VC 3 to $J = \mathsf{true}$, and VC 4 to the ordinary Join Lemma — recovering the unconditioned methodology.

12 The metatheorem

Theorem 12.1 (Soundness — conditioned VCs ⇒ RA-linearizability)**.** *Any MRDT satisfying VCs 1–4 is RA-linearizable (Definition 3.2, read up to* $\approx$*).*

The proof is generic — it never inspects a particular datatype — and factors through two lemmas, each proved over the abstract signature with $\approx$ an arbitrary congruence for do and merge.

Remark 12.2 (Route actually taken)**.** *The convergence lemma below is stated via the swap-based VC 3 for exposition. That route is machine-refuted for the tombstone-free RGA; convergence is mechanized by direct canonical-state characterization instead. See Section 14.*

Lemma 12.3 (Convergence)**.** *VCs 1, 2, 3 imply that all* lo^E*-respecting admissible enumerations of a backward-closed reachable* E *fold from* σ_0 *to* $\approx$*-equal states; hence* $\sigma^*(E)$ *is well defined up to* $\approx$*.*

Proof. Any two lo^E-respecting enumerations of E are related by repeatedly transposing adjacent lo^E-incomparable events. Because lo^E is non-transitive, "respecting" is the elementwise `Pairwise` condition rather than sortedness, so this order-repair is more delicate than for a total order — it is exactly what the generic engine `eq_convergence` carries out, and we appeal to it rather than re-derive it. Each transposition preserves the fold up to $\approx$ by VC 2, provided the swap precondition holds at the transposition point; VC 3 supplies it along the whole enumeration (base at each event's enabling fold, preserved by each step, scoped to the enabled events — the only ones the repair transposes). The engine consumes only that $\approx$ is an equivalence, do is $\approx$-congruent, and a swap oracle of the shape VC 2+3 provides (Lean: `eq_convergence`, kernel-clean). □

Lemma 12.4 (Canonical adequacy)**.** *VC 4 and Lemma 12.3 imply that every reachable version state is* $\approx\ \sigma^*$ *of its delivered event set.*

Proof. Induction over the version DAG. Initial: the empty fold. Update node: appends one delivered operation to a canonical fold (definitional). Merge node with LCA l and branches a, b: by hypothesis $a \approx \sigma^*(E_a)$ and $b \approx \sigma^*(E_b)$, and $l \approx \sigma^*(E_l)$. First rewrite a, b to literal canonical folds using $\approx$-congruence of merge (so VC 4, stated for branch folds, applies); then VC 4 gives $\mathsf{merge}\, l\, a\, b \approx \mathsf{fold}\, l\, \pi$ for a lo^E-respecting π of the branch events $E_a \cup E_b$. Now E_l causally precedes those branch events (they were generated on states derived from l), so a lo^E-respecting enumeration of $E_l \cup E_a \cup E_b$ factors as (an lo^E-enumeration of E_l) followed by π; hence $\mathsf{fold}\, l\, \pi = \mathsf{fold}\, (\sigma^*(E_l))\, \pi \approx \sigma^*(E_l \cup E_a \cup E_b)$ by Lemma 12.3 (the engine is start-state generic). Backward closure keeps every enumeration admissible. □

Proof of Theorem 12.1. Immediate from Lemma 12.4: each version state $\approx \sigma^*(E_v)$, which is Definition 3.2. (Unconditioned template: `ra_linearizable3_of_joinC` plus the closure-indexed adequacy bridge, already kernel-clean with nine instances; the conditioned metatheorem generalizes it by carrying $\mathsf{Inv}/\mathsf{app}$ through the two lemmas.) □

Corollary 12.5 (Strong eventual consistency)**.** *Two versions with the same delivered set* E *have* $\approx$ *states: both* $\approx \sigma^*(E)$ *by Theorem 12.1, hence* $\approx$ *each other.*

13 The tombstone-free RGA discharges the conditioned VCs

The RGA is a forest of identified nodes; do inserts a fresh node under a resolved anchor or physically deletes a node and rehomes its children; merge keeps OR-set survivors and re-anchors each by climbing to the nearest surviving ancestor. $\mathsf{resolve}\, \sigma\, p$ returns the first live id on a recorded path p; $\mathsf{anc}\, \sigma\, x$ is x's current parent. Take $\mathsf{Inv} := \mathsf{wf} \wedge \mathsf{id\_mono} \wedge$ (root not reinserted) and $\mathsf{app} := \mathsf{accurate} \wedge \mathsf{fresh}$. This section presents the discharge as it was *first attacked*: swap-based, with the two hard VCs reduced to one faithfulness invariant.

First, the evidence that the flat schema is out of reach. The introduction's claim that this data type is *provably outside* the flat VC set rests on two kernel-checked refutations at the do level, one per escape route (the visual account, with the traces drawn, is

`Sal/MRDTs/RGA/doc/why-the-path-matters.pdf`, §6). *Route one: no path.* The path-free splice predecessor (`RGA_Splice`, branch `wip/rga-splice`) has $\mathsf{Ins}(t \text{ after } x)$ and $\mathsf{Del}(x)$ genuinely non-commuting (the splice destroys the parent pointer), so the flat schema's commutation iff (`rc_non_comm`) forces the entry $\mathsf{rc}(\mathsf{Ins}(t \text{ after } x), \mathsf{Del}(x)) = \mathsf{Fst\_then\_snd}$ — and a non-Either entry awakens the conditional-commutation base VC (`cond_comm_base`: an rc-ordered pair must be order-insensitive at application whenever a third op also conflicts), which the three-operation trace $\mathsf{Ins}(10 \text{ after } 5); \mathsf{Del}(5); \mathsf{Ins}(12 \text{ after } 5)$ from $\{(5, \cdot, 0)\}$ refutes: the first insert is rehomed onto the root in the mandated order but dangles in the flipped one (`cond_comm_base_violated`, by evaluation). *Route two: the path.* Carrying the ancestor path repairs exactly that trace (`trio_converges`) but the repaired commutation is *reachability-conditioned*: the mechanized `rc_non_comm'` assumes the carried paths are `accurate` in the state, and dropping that hypothesis is refuted by a lying record — $\mathsf{Ins}(10 \text{ after } 5, \langle 7 \rangle)$ accurate against $\mathsf{Del}(5, \langle\rangle)$ claiming 5 hangs off the root; whichever runs second applies *its own* record and the orders diverge (`inconsistent_diverges`). Accuracy is a property of the execution that generated the operation, not of the state it meets, so no $\forall\sigma$ equation can carry it: the schema cannot be satisfied, only *changed* — which, done soundly, is the conditioned framework of §10.

Definition 13.1 (Recorded-path faithfulness RecPathFaithful)**.** *RecPathFaithful $a\,\sigma$: the recorded path $\mathsf{rec}\,a$ is a live subchain of genuine ancestors of a's target in σ — true* anc*-ancestors, nearest first, consecutive pairs real parent edges, and exactly the chain live at a's generation. (This is `accurate/HistFaithful` promoted to a reachability invariant.)*

Lemma 13.2 (RecPathFaithful is a reachability invariant)**.** *RecPathFaithful $a\,\sigma$ holds at every reachable σ in which a is applied.*

Proof. Induction on the operations building σ. *Birth:* do sets $\mathsf{rec}\,a =$ resolve-path at generation = the live-ancestor chain, so RecPathFaithful holds (`chainFaithful_of_histFaithful`). *Later* Ins *with fresh id f:* attaches f as a descendant — no reparenting, no liveness change for prior events' chains. f cannot appear spuriously in any $\mathsf{rec}\,a$: its members are older ids $< a.\mathsf{id} \leq$ every id extant at a's birth, whereas f exceeds all of them, so $f \notin \mathsf{rec}\,a$ definitionally. The resurrection/clash regime (`chainFaithful_not_preserved_under_clash_ins`) is therefore *unreachable* by freshness (`fresh_not_mem_recList`, `clash_recList_excluded`), not excluded by a side condition. *Later* $\mathsf{Del}\,x$*:* flips x's liveness only; $\mathsf{rec}\,a$'s entries stay genuine ancestors, one possibly now dead — still a live subchain (`chainFaithful_doDel_faithful`). □

Lemma 13.3 (Key Lemma: resolution over a live subchain)**.** *For any live subchain of genuine ancestors pre of x, in every reachable σ,* $\mathsf{resolve}\,\sigma\,pre = \mathsf{anc}\,\sigma\,x$*. (Lean: `subchain_resolve`, `RGA_SubchainResolve.lean`, kernel-clean.)*

Proof. resolve returns the first entry live in σ; each entry is a genuine ancestor, so this is the nearest live ancestor among *pre*. It is the nearest overall: no live ancestor $c \notin pre$ is nearer than that first-live entry — (a) if c was live at capture it was in the captured live chain, so $c \in pre$ (and any *pre*-entry nearer than the first-live one is dead in σ, hence not c); (b) if c was dead at capture it is still dead (physical, tombstone-free deletion is permanent); (c) an event inserted after capture attaches as a descendant, never a new ancestor of the pre-existing x. The mechanization absorbs these three cases into one preserved invariant $\mathsf{LiveChain}\,\sigma\,x\,pre := (\text{root unstored}) \wedge \mathsf{live}\,\sigma\,x \wedge \mathsf{IsAncPath}\,\sigma\,x\,(pre.\mathsf{filter\,live})$, which holds at capture, is preserved by each insert (`isAncPath_upd`) and delete (`isAncPath_surgery`, chain splicing across a deleted entry), and yields the conclusion. Case (b) is discharged by an explicit

hypothesis $t \notin pre$ on inserts (`okStep`) — the mechanized form of "no resurrection," itself a consequence of monotone allocation, every *pre*-entry sitting below x's id. This replaces the earlier full-chain lemma `hres_of_lchain`, which failed on the proper subchains rehoming produces. □

Proposition 13.4 (RGA discharge)**.** *The RGA satisfies VCs 1–4. (Proof sketch below along the swap route; in the final mechanization the entire route — the swap VC, the* ChainFaithful *threading, the per-id* BranchInv *— is superseded by the canonical-state engine of Section 14, and only the Key Lemma survives; see the audit note there.)*

Proof. *VC 1* (discipline): Inv preservation and the generation model are `ConditionedConfiguration` instantiated at the RGA, with the RGA's do-invariance lemmas; `conditioned_premises` supplies the timestamp/freshness facts. *VC 2* (swap): `general_swap_bothFaithful` / `eqSwap_of_bothFaithful` — concurrent operations commute up to $\approx$ when both are *faithful* (neither need be exact/accurate), kernel-clean. *VC 3* (reorder invariant): $J :=$ ChainFaithful on the enabled scope, with steps `chainFaithful_incompStep`/`incompFold`; its *base* — each event faithful at its enabling fold — is exactly RecPathFaithful (Lemma 13.2) projected through the Key Lemma (Lemma 13.3): the residual goal $\mathsf{resolve}\,\sigma\,((\mathsf{rec}\,a) \setminus t') = anch'$ when an ancestor t' becomes leftmost live is the Key Lemma on the remaining subchain. *VC 4* (Join): per-id extensional match (`eq_merge2_of_branchInv2`); the anchor-coincidence clause is the single-sided I_4 (`eq_merge_single`, kernel-clean) and, cross-branch, the Key Lemma directly (`branchInv_doDel_crossBranch`, kernel-clean); order-independence is Lemma 12.3 at start state l. (In the final assembly this per-id BranchInv threading is itself superseded by the birth-anchor bridges of Section 14, which need no invariant threading at all.) Both hard VCs (3 base, 4 cross-branch) thus rest on RecPathFaithful and the Key Lemma — proved by construction from freshness and permanence. □

Corollary 13.5. *The tombstone-free RGA is RA-linearizable (Theorem 12.1 + Proposition 13.4), and strongly eventually consistent (Corollary 12.5). To our knowledge this is the first mechanized RA-linearizability / SEC result for an RGA with physical, tombstone-free deletion (cf. Gomes et al., OOPSLA'17, for the tombstoned RGA).*

Remark 13.6 (Other instances)**.** *The nine flat production MRDTs (counter, OR-set, MVR, …) take* $\mathsf{Inv} = \mathsf{app} = \mathsf{true}$*: VC 2 is their unconditioned commutation, VC 3 is* $J = \mathsf{true}$*, and VC 4 is the ordinary Join Lemma — already discharged (`ra_linearizable3_of_joinC`). They inherit RA-linearizability from the same Theorem 12.1. The RGA is the instance that exercises every conditioned mechanism; the framework supplies the generality, the RGA the novelty.*

14 How it was actually mechanized (and two dead ends)

The exposition above (Sections 7–**??**) is the *intended* route, and it correctly frames the contribution: conditioned VCs ⇒ RA-linearizability, RGA as instance. But the mechanization did *not* follow the swap-based convergence of VC 3. That route was tried and **machine-refuted twice**; convergence was then proved by a different technique. We record the actual route honestly — the refutations are themselves findings about why tombstone-free deletion is hard.

A postscript, established after the assembly was complete: a kernel-level dependency audit of the capstone's compiled proof term shows the subsumption is *total*. The final theorem's dependency cone contains no swap lemma, no ChainFaithful/RecPathFaithful threading, and no BranchInv — of the swap route only the Key Lemma remains load-bearing. The route's files stay proved and kernel-clean, retired under `Development/`; of the investigation's ∼17k lines, the

capstone's dependency cone spans ∼8k. The lesson generalizes the refutations: for a datatype whose operations reference mutable structure, the set-indexed characterization is not just an alternative to linearize-and-swap — it makes the swap layer unnecessary even where the swaps are provable.

Two machine-checked refutations of the swap route. The reorder-invariant convergence (VC 3, "thread faithfulness through adjacent swaps") fails for the tombstone-free RGA because *rehoming makes an event's recorded chain time-relative*: correct at generation, wrong at reordered intermediate states.

- lo^E **is not transitive** (`RGA_DepComp_Gate`, kernel-clean refutation). Reordering dependencies-contiguous-first needs transitivity of the linearization order; a physical delete severs an operation's applicability-dependence on an ancestor ($b \to a \to o$ but $\neg(b \to o)$ after a's target is deleted). Witness: $l = 0 \leftarrow 1 \leftarrow 2 \leftarrow 3$, delete node 1, delete node 2.
- **In-place threading fails at the ancestor-insert step** (`RGA_SimulInduction2`, kernel-clean refutation). Even threading without reordering, an event's recorded chain can be *ahead* of the state (it anticipates a delete not yet folded), so ChainFaithful of the recorded chain is false at that intermediate fold although it heals later.

Both say the same thing: the linearize-and-swap method, which needs faithfulness at every *intermediate* reordered state, is the wrong vehicle. This is exactly what tombstones prevent (Gomes et al. keep deleted nodes so dependencies never rewire); the tombstone-free RGA rewires them.

The technique that works: direct canonical-state characterization. Instead of swaps, characterize the fold state as a pure function of the applied event *set* — the update analog of the merge bridge (which always worked). For a finite applied set F, define $\mathsf{survivors}\, F$ (OR-set survival) and $\mathsf{canonAnc}\, F\, k$ (climb k's recorded chain to the nearest survivor of F); let $\mathsf{CanonMatch}\, F\, s$ say s matches this per id.

Lemma 14.1 (Canonical fold)**.** *Along any* lo^E*-respecting enumeration* π*,* CanonMatch (π*-applied*) (fold $\sigma_0\, \pi$) *(Lean:* `canon_fold`*, kernel-clean). Two folds of the same* E *then both match* CanonMatch E*, hence are* $\approx$ *— convergence with no swaps and no intermediate-state faithfulness (*`RGA_update_convergence_canc` *discharged from the per-event generation discipline* GenDisc2C *in* `RGA_update_convergence_final`*).*

The invariant is over the *applied* set, so $\mathsf{canonAnc}\, F$ never anticipates a future delete — exactly why it succeeds where the swap route (which reasons about not-yet-applied events) fails. The Key Lemma (Lemma 13.3) survives unchanged as the per-id climb-vs-resolve coincidence; RecPathFaithful is subsumed by CanonMatch's per-survivor `LiveChain`.

The metatheorem as a generic $\approx$-quotient functor. Theorem 12.1 is mechanized generically over `ConditionedMRDTSig` (`RA_linearizable_up_to_eq`, `GenericEqQuotient`, kernel-clean). It quotients the state by $\approx$ over the Inv-subtype and transfers the datatype's $\approx$-Join to the structural-$=$ template `ra_linearizable3_of_joinC` by `Quotient.sound`; the readback is over *raw* fold, matching the RGA's convergence. The datatype supplies five VCs: EqEquiv, InvPres, CongVC (update/merge/query $\approx$-congruent on Inv), InvInvVC, and the $\approx$-Join EqJoinLemma3C. Two subtleties the mechanization forced, each machine-verified:

- **The merge $\approx$-congruence is false on the full state type** (`merge_eq_congr_l_fails`, axiom-free), true on the reachable subfamily ($\mathsf{wf} \wedge \mathsf{id\_mono} \wedge$ forest): the LCA-climb reads

anc l off domain l, where $\approx$ is silent. Hence the Inv-conditioning of CongVC is load-bearing, and the quotient is over the Inv-subtype (`merge_eq_congr_inv`).

- **Invariant preservation needs well-formedness, not applicability.** InvPres cannot demand unconditional preservation (do at a bad id breaks the invariant), nor full app (which the execution model does not guarantee — `Step3.apply` applies with only timestamp freshness). The honest condition is a datatype predicate WfOp (for the RGA: fresh id for Ins; resolve $s\,p \neq x$ for Del $p\,x$), preserved by the real do (`rgaInv_doOp_fresh`, accuracy *removed*) and holding along reachable folds (WfOpReachable). A totality guard on the lifted step is provably transparent on the execution domain (applySeqW = applySeq); the exposed theorem is over the real do.

The completed assembly. Everything after the quotient functor was, at the first writing of this note, "bounded assembly." Carrying it out surfaced three further machine-checked refutations that reshaped the target and the witness discipline; we record them because each is a fact about tombstone-free deletion, not about our proof.

- **At a merge union, applicability-tolerant witnesses do not exist.** An LCA-first enumeration pre-applies one branch's anchor-killing deletes before replaying the other branch's inserts, staling their recorded paths — so even *no-op-feasible* witnesses (each step applicable or a no-op) are unsatisfiable from the initial state (`RGA_HEnum_Refutation`). Worse, on a seven-event two-branch execution *no* born-applicable enumeration of the union replays both branches' recorded observations: rehoming makes the *order of concurrent deletes* observable in survivors' stored parents, and the branches observe it differently. Consequently the linearization target itself must be the *raw* do-fold, up to $\approx$ — any guarded or feasibility-filtered fold is unsatisfiable at merge unions (`IsRALinearizable3Eq`, `GoodConfig3H.lean`).
- **The witness discipline is the engine's own, plus payload honesty.** Each version carries, existentially, an enumeration satisfying the canonical engine's per-step discipline (`CanonFoldOK`: fresh nonzero ids, no id reuse, live recorded chains) *and* payload honesty (`HonestPayloads`: delete targets and recorded-chain entries are root-or-inserted). The second clause is forced by a counterexample: the fold discipline alone admits a dead-target delete (a no-op), and a later insert reusing that recorded dead id breaks the no-id-reuse clauses — honesty of *payloads* is set-level, hence permutation-invariant, and it is exactly what extension at a fresh apply needs (`rga_hHext_discharged`). The discipline *joins* at merges with the union witness being the LCA-then-delta concatenation $\rho_0 +\!\!+ \pi_0$ itself: no reordering, no swaps, anywhere (`rgaJoinH_of_canon`).
- **The generation discipline is derived, not assumed.** Every event is accurate at the fold of its *transitive dependencies* (`GenDisc2C`), provable from *born accuracy* — the op was generated against a causal fold of the events its replica had seen (`genDisc2C_of_born`). Merge canonicity then reduces to three leaves, all discharged: σ_0 id-monotonicity with *no fold induction* (the stored anchor is canonAnc of the recorded chain, whose entries are dependencies, which are Lamport-below — `rga_Hdec_discharged`); a per-id causal set-algebra whose only non-trivial clauses are dependency-fold provenance (`rga_hcaus_discharged`); and per-survivor *birth-anchor bridges* reducing the merged forest's anchors to record coherence at dependency folds, where the generation discipline makes every recorded chain live (`rga_hbridge_discharged`, `hin_of_genDisc`).

The residual, quantified once over the execution, is a single per-step assumption (`HonestDelivery`):

born accuracy plus *born-applicable delivery*. It is genuinely irreducible — it is the generation discipline tombstone-freedom forces — and it is also all there is: Lamport clocks, timestamp uniqueness and visibility-support are structural fields of the configuration (dishonest-clock executions are unrepresentable); nonzero insert ids and nonzero delete targets follow from the delivered op's own wellformedness at a representative of its head class; nonzero delete *timestamps* are Lamport-derived from the target's insert. The end-to-end theorem (`rga_RA_linearizable_honest`; mainline entry point `rga_tombstone_free_ra_linearizable3_eq`, file `MRDT_Instances/RGA_TombstoneFree/RA_Li` closes Proposition 13.4 and the corollary unconditionally modulo that assumption.

A second conditioned instance: the bounded counter — conditioning for safety. The tombstone-free RGA needs conditioning to get *convergence*. The bounded counter (per-replica escrow: grow-only tallies ($\mathsf{incs}, \mathsf{decs}$) per replica, per-slot group merge; mirror of `Sal/CRDTs/Bounded_Coun` shows the same three mechanisms delivering *safety* for a datatype whose convergence is flat: all its operations commute, the eight VCs discharge along the counter route, and the catalogue capstone (`BC_ra_linearizable3_eq`) is an identity instantiation. What the flat theory cannot even state is the property the datatype is named after. Its CRDT development is explicit: "the bound is enforced operationally by well-behaved clients" — a client refuses to issue Dec without slack — and that discipline lives outside the formal development. Conditioned, it moves inside: Inv is the per-replica escrow invariant ($0 \le \mathsf{decs}\, r \le \mathsf{incs}\, r$), app is the client's check (a Dec needs slack in the *issuing replica's own* slots — positive, and checkable against the issuing replica's state), and the contract on executions (`BCHonest`: every decrement was applicable at the fold of its causal past) is the counter's `HonestDelivery`. The theorem (`bc_version_inv`, kernel-clean): in every reachable configuration with honest history, *every version — heads, LCAs, everything the store registered — satisfies the invariant*; hence the counter's value is non-negative everywhere (`bc_value_nonneg`). The proof is a counting argument riding the flat machinery: every version's state is a fold of its event set (canonicity), fold slots *are* per-replica event counts (the group merge is inclusion–exclusion on counts, via the LCA lemma), and the vis-maximal decrement's honesty at its own causal past — which lies inside the version by causal closure — bounds the decrement count by the increment count. At ∼600 lines against the RGA's ∼8k, it is also the first calibration point for which conditioning machinery is generic: the same invariant-at-generation shape, none of the positional structure.

A third conditioned instance: the mergeable queue — when no $\xrightarrow{\mathsf{rc}}$ can exist. The queue of the certified-MRDT line of work (Soundarapandian et al., PLDI'22, *Certified Mergeable Replicated Data Types*) is the catalogue's first data type whose conflict graph is a *clique* rather than a star: two concurrent enqueues genuinely do not commute (queue order is arrival order), so VC 1's iff forces an $\xrightarrow{\mathsf{rc}}$-edge between them, and any orientation of a self-conflicting class composes with itself into a length-2 path — every candidate assignment violates `no_rc_chain` (Open Question 15.7). The flat engine is structurally unavailable, and the instance goes through the framework's per-configuration join hook instead (`JoinLemma3At`, `goodConfig3_merge_at`): prove the Join Lemma directly, at every reachable configuration.

The implementation is the functional queue with Peepul's merge, verbatim: the state is a list of $(tag, value)$ pairs, head first; $\mathsf{enq}\, v$ appends a fresh element tagged with the operation's own timestamp; $\mathsf{deq}\, t$ removes the element tagged t, wherever it sits; and the three-way merge keeps the LCA's elements that survive in both branches (in LCA order), then each branch's new arrivals (in branch order). The client-facing API is unchanged — dequeue() at a replica reads

its local head — but the *operation* records which element that call removed: app for deq t is "t is the head of the issuing replica's state". As with the RGA, the generation discipline is the datatype's natural client behaviour, captured rather than imposed.

The proof exhibits Peepul's merge as its own certificate. The Join Lemma asks for a delivery-respecting enumeration of $E_1 \cup E_2$ whose fold is $\mathsf{mergeL}(\sigma_0, \sigma_1, \sigma_2)$; the witness is the concatenation $\rho_0 \cdot (\rho_1 \setminus E_0) \cdot (\rho_2 \setminus E_0)$ of the given enumerations — exactly the merge's shape, no reordering. Three facts make the fold of that concatenation compute the merge, all consequences of the closure premise (branch event sets are backward-closed under $\xrightarrow{\mathsf{vis}}$ among non-commuting pairs), timestamp uniqueness, and an honest-history contract (`QHonest`: every dequeue names a tag whose enqueue is $\xrightarrow{\mathsf{vis}}$-prior — the queue's `HonestDelivery`, discharged by the app head-check via `qHonest_of_applicable`): a fold from the empty queue is "the enqueues, in order, minus the dequeued tags" (the fold formula, which needs honesty — a dequeue preceding its enqueue would consume nothing); a delta's tags are fresh with respect to the LCA (uniqueness); and a cross-branch dequeue is impossible — a deq t in one branch pulls its enqueue into that branch by closure, so if the enqueue of t is the *other* branch's delta, it would lie in both branches, hence in the LCA, contradicting delta-ness. An LCA element then survives the union iff it survives in both branches, which is precisely the merge's first clause.

The payoff is the specification. In the PLDI'22 development the queue was the hardest case exactly because its specification had to be written in a bespoke relational language over event partial orders, with an explicit clause recovering *which element is the head*; the merge was then verified against that ad-hoc spec. Here the specification is the same generic statement as every other instance in Table 1 — every version is the fold of a delivery-respecting linearization of its event set — and head-ness, FIFO order, and once-per-element consumption are *read off* the fold rather than specified. (One semantic caveat, present equally in Peepul: two replicas that concurrently dequeue the same head both consume that one element — the merged state removes it once. RA-linearizability makes this visible instead of hiding it: both deq t events fold over the single enq t.) At ∼870 lines (`MRDT_Instances/MergeableQueue/`), the instance sits between the bounded counter's ∼600 and the RGA's ∼8k — positional structure but no rehoming.

The factorization: one metatheorem, three join species. With three conditioned instances on the books, the common structure is a theorem rather than a convention. All three developments ran the same induction — thread a per-configuration honesty contract through reachability, invoke a Join Lemma at each merge — and that induction is now extracted once (`Metatheory/HonestReach.lean`): `HonestReach` D H is LTS reachability where every step leaves a configuration satisfying the contract H, and the metatheorem (`ra_linearizable3_of_honest_reach`; axioms `propext`, `Quot.sound`) states that if every H-configuration admits the Join at its core (`JoinLemma3At`), every H-honestly reachable configuration is per-version RA-linearizable. Plain reachability is the degenerate contract $H = \top$, recovering the unconditional bridge. The queue's and the bounded counter's bespoke inductions are now one-line corollaries; the RGA's honest-delivery induction is the same shape under its $\approx$-quotient.

What does *not* factorize is the join discharge, and that is the finding: the three instances discharge `JoinLemma3At` by three genuinely different mechanisms — *conditioned commutation* (the RGA: operations commute on invariant-satisfying states), *flat commutation* (the bounded counter: the CD route, contract $\top$), and *direct witness* (the queue: the merge is its own lin-

earization certificate). Conditioning thus splits as

$$\text{generic honest-reachability metatheorem} \times \text{per-datatype join discharge},$$

and the framework's job at the boundary is to *receive* a join, not to produce one. One further regularity is worth recording: all three honesty contracts have the shape "P holds of each event at the fold of its causal past" — born accuracy and applicable delivery for the RGA, slack for the counter, head-ness for the queue — with P the signature's client-checkable `app` (or its accuracy analogue). That shape is extracted once (`GenHonest`, with `AppHonest` its `app` instance and `ra_linearizable3_of_genHonest_reach` the composite metatheorem; `Metatheory/GenHonest.lean`), and the counter's and the queue's contracts are re-derived as instantiations (`BCHonest_iff_genHonest`, `qHonest_of_genHonest`). A generic *safety* metatheorem completes the factorization; it is the subject of the next paragraph, and the counter's counting argument indeed turned out to mark where the per-datatype content sits.

The generic safety metatheorem: causal witnesses, and an escrow route. The safety half of conditioning also factorizes, but the obvious formulation is *false*, and the refutation comes from the flagship safety instance itself. One would like: if updates preserve Inv on applicable operations and the history is honest, then Inv holds at every version — proved by inducting along the version's canonical enumeration. But canonicity constrains the enumeration only by lo, and for the bounded counter — all of whose operations commute — lo is *empty*: [dec, inc] is a perfectly canonical enumeration of an honest version, and its intermediate state violates the escrow invariant. Only *endpoints* of canonical folds are guaranteed; commuting causal predecessors may float behind the event that needs them. The repair is to restrict the witness class, not the induction: a *causal* witness additionally linearizes $\xrightarrow{\mathsf{vis}}$ (`CausalCanonical`), and its prefixes are exactly the vis-closed, future-free sets containing each event's whole causal past. Over those, the metatheorem holds (`version_inv_of_causal_canonical`; axioms `propext`, `Quot.sound`): Inv at every version, from honesty (`HonestApp`: `app` at some causal fold of the event's causal past — what the issuer's own state witnesses) and one per-datatype obligation `SafetyStep` (applicability transfers from the causal past to any admissible prefix, fused with Inv-preservation). For datatypes whose operations all commute the causal witness is free (`causalCanonical_of_all_comm_rc_either`, a pointwise permutation argument). The bounded counter discharges `SafetyStep` in three lines — extras in a prefix are other replicas' events and cannot touch the issuer's slots — and `bc_version_inv` is re-derived as a corollary, retiring the bespoke vis-maximal counting apparatus, which is thereby revealed as compensation for inducting over non-causal witnesses. A second, independent route generalizes that retired argument itself: for datatypes *measured* by affine observations (fold values = event-set counts), an *escrow metatheorem* (`escrow_version_inv`) yields the same conclusions with no causal witness and no Inv-preservation at all; `bc_version_inv` is derived a second time this way. The queue, by contrast, *refuses* `SafetyStep` — "t is the head" is not stable under a concurrent honest dequeue of the same head, and correctly so: the double-dequeue execution satisfies any sound obligation, and the queue's stable residue is event-level (`QHonest`), consumed by convergence rather than by a state invariant. The full analysis, including two further refuted formulations, is the pen-and-paper memo `Development/GENERIC_SAFETY_PENPAPER.md` — written, per this project's standing lesson, before any Lean. (A natural follow-up is a *conditioned step relation* — apply guarded by `app` at the head state — under which honesty becomes a theorem of the LTS rather than a contract; deferred.)

A small fourth instance: the FWW reservation register — payload arbitration, and what honesty cannot buy. A register claimable once (a seat, a username): sequentially the first claim wins; concurrently, the winner is the claim that is first in *Lamport* time. This is the positive complement to the metadata-free kill-test (`lww_merge_needs_timestamps`): arbitration whose key is not in the state is impossible, and arbitration whose key *is* in the state is a semilattice triviality — the state is the minimum claim $(\mathsf{ts}, \mathsf{rep}, v)$ under the lexicographic order (or unset), update and merge are min, and the entire eight-VC discharge is min-algebra (∼300 lines, the smallest conditioned instance). The characterization theorem (`fww_version_min`) is honesty-free — semilattice folds are enumeration-independent — and says: at every version of every reachable configuration the register holds exactly the minimum-timestamp claim of its event set, unset exactly on the empty set. Since timestamps respect causality, a causally later claim never displaces an installed one; the min rule arbitrates only among *concurrent* claimants, and it must, on pain of divergence. The instructive part is the generation discipline: app for a claim is "the register is unset in my causal past", and *deliberately no theorem consumes it* — two honest concurrent claimants both see unset, both claim, and each locally wins until a merge disabuses one. "Unset" is the minimal example of an applicability check that is *not stable under concurrent honest extension* (contrast the escrow counter's own-slot slack, which is — precisely the boundary the safety metatheorem's `SafetyStep` draws). A merge-based register is a reservation, confirmed only at causal stability; it is never a mutex.

The composition payoff: tombstone-free Peritext in a thousand lines. The composition kit's first real consumer is the instance that motivated it: tombstone-free rich text, built as RGA⊗marks (`MRDT_Instances/Peritext_Composed/`, capstone `peritextComposed_ra_linearizable_up_to_e` every version of every reachable configuration RA-linearizable up to $\approx_{\mathrm{RGA}} \times =$, under precisely the RGA's own honest-delivery premise read through the first projection — character operations guarded, mark operations unguarded). The mark store is an `ORSetCore` instantiation (81 lines); marks carry their endpoints' recorded paths, and resolution happens at *read* time by the RGA's own path-climbing — so no update or merge crosses components, the coupling lives in the query, and the one obligation read-coupling costs is discharged kernel-clean (`peritextRender_congr`: ≈-related states render identically, from the RGA's resolution reading only live). The cost center was exactly where the analysis predicted: re-running the RGA's honest-delivery induction over the product LTS (790 lines, the component-2 steps entering as stutters), consuming the RGA chain's content lemmas as certificates, re-proving none of them. Total: 1,064 lines against a from-scratch alternative assessed structurally infeasible — every lemma of the ∼10k-line RGA chain would have acquired mark cases. Composition is no longer a conjecture in this catalogue; it is how the newest instance was built.

One boundary is worth flagging, because it extends the decorative-vs-load-bearing taxonomy of app (§15, the FWW register and the budget cart) to the compositional setting. The composite's honesty premise (`PeritextHonestDelivery`) guards only the *character* operations — the mark component's operations are unconstrained. Marks carry the same data a discipline would need (each endpoint is a character id plus its recorded ancestor path), so it is natural to ask whether *mark-anchor honesty* — the mark-side analogue of the RGA's born accuracy — composes. It does (`MRDT_Instances/Peritext_Composed/MarkHonesty.lean`), and the way it does is the finding. The product *signature* cannot express it: its app is componentwise $(\mathsf{app}^{\otimes}(\mathsf{inr}\, o)\, s = \mathsf{app}_2\, o\, s.2)$, structurally blind to the other component. But the honesty *premise* is a free-standing predicate over the product LTS — exactly as the RGA's own born

accuracy is — and there `MarkAccurate` reads the character component of a mark's causal-past fold freely; it respects the character quotient (`markAccurate_congr_1`), so it is a well-defined predicate on the very quotient the capstone speaks about; and the strengthened contract still yields the capstone (`peritextTF_ra_linearizable_honest`) unchanged, because the linearisation proof consumes only the character clause — marks converge and linearise as an OR-set regardless. The mark clause is thus *decorative for linearisability and load-bearing only for the read layer*, and the read-layer facts it buys are honest but weak (`MarkIntent.lean`): resolution stays within a mark's recorded chain (`mark_start_in_recorded`), a non-root endpoint is live (`mark_start_live`), and under accuracy the endpoint is an issue-time ancestor of its anchor (`mark_start_within_recorded_ancestry`).

That last fact is a *containment* bound, not the "formatting does not leak" guarantee an earlier draft of this note claimed — and the difference is instructive. The paper keeps deleted characters as tombstones, so a mark boundary anchored to a deleted character does not move. Our sequence is tombstone-free, so the boundary is recovered by climbing its frozen recorded path, and `resolve` climbs *tree ancestry* — which is *earlier* in reading order and can skip surviving siblings. So deleting a mark's anchor migrates the boundary *backward* in the document: formatting leaks onto text that was never in the span. The theorem bounds this drift to the recorded ancestor chain (no wild jump to unrelated text) but does not prevent it; the frozen-path design simply does not have the paper's positional semantics. Correct tombstone-free positioning needs *document-order* rehoming — the boundary moving to its nearest surviving neighbour in reading order — which the fused boundary-node variant supplies (boundaries are RGA nodes, rehomed to preserve reading order) at the cost of atomicity; this is the Peritext read-model work that remains (Open Question 15.10).

The methodological point is worth stating, because it is why the overclaim survived a machine-checked proof. **RA-linearisability here certifies convergence *to the datatype's own sequential fold*, not correctness of that fold.** Classical linearisability checks a concurrent history against a *separate* sequential specification; our fold *is* the reference, so a defect that lives in `do` itself — a wrong rehoming — appears identically in the merged state and in its sequential-fold "spec," and linearisability faithfully certifies that the two agree, on the wrong answer. Single-replica semantic bugs are therefore invisible to it; only an *independent* intent specification (the paper's §A.2 examples; our read-side theorems) can catch them — which is exactly why Peritext separates convergence from intent preservation, and exactly the obligation `MarkIntent` was meant to discharge and (in its leak-prevention form) does not.

The composition lesson still lifts cleanly: a component's honesty discipline is load-bearing exactly where the *coupled read* depends on it, not where convergence does — and the read-coupled boundary carries honesty precisely because the honesty predicate lives free of the signature's componentwise `app`.

15 Open questions

The mechanization settles what the paper claimed; it also sharpens what remains genuinely open. Ten questions, roughly in order of theoretical depth:

Open Question 15.1 (Is there a rely-guarantee program logic for MRDTs?)**.** *The framework already carries the skeleton of a verification-condition logic: the signature-plus-conditioning is the "program", RA-linearizability up to $\approx$ is the judgment, the eight VCs are local proof obligations, the metatheorem is their soundness theorem, and the product $\otimes$ with* `JoinLemma3At`

is a frame/composition rule (the coupling hierarchy L0–L3 stratifying it by interaction strength, as disjoint-vs-shared reasoning does in separation logic). The conditioning layer is specifically rely-guarantee: honest delivery is the rely (an assumption on other replicas' and clients' steps), convergence/safety up to $\approx$ is the guarantee, merges are environment steps, and the "stable under concurrent honest extension" condition the safety metatheorem turns on (§15 below — the bounded counter succeeds and BudgetCart/FWW fail on exactly it) is the rely-guarantee stability side-condition, arrived at independently. What is missing to make it a program logic proper is a syntax: a language building do/merge *from primitives (a per-replica counter cell, OR-set membership, an LWW/FWW* max/min *cell, a sequence node) and combinators (the product $\otimes$, the indexed map, the payload-parametric wrap of* `ORSetCore`*), each carrying its local VC contribution and its join-species, such that well-formedness entails RA-linearizability by construction — so that writing an MRDT and proving it converge become one act, as separation logic made heap-program verification syntax-directed. The semantic combinators already exist and are each separately proven sound; what is open is the grammar together with a single soundness theorem for the grammar (not one per combinator). This question subsumes the two framework-theory questions below:* CDVC3*-derivability (15.2) asks whether the rule set is minimal, and the structure theorem asks to classify the primitives.*

Open Question 15.2 (CDVC3-derivability)**.** *Is* CDVC3 *derivable from VCs 1–7 (plus the unconditional delta laws where they hold)? Specializing the merge to an LCA-blind lattice join — the CRDT case — the question closes exactly: there, the corresponding causal-delta bound is equivalent to the Join Lemma and no strictly weaker bridge VC exists, with both attack routes characterized — the positive induction is circular at two identified case shapes, and the countermodel space is narrowed to non-atomic lattice states (all of this mechanized, in* `Sal/CRDTs/Metatheory/`*). The ternary question, by contrast, is now* ***closed****:* `CDVC3` *is an independent rule of the ternary bundle (*`cdvc3_not_derivable_from_core_delta`*,* `Refutations/CD_Not_Derivable_Ternary.lean`*, kernel-clean) — there is a* `ConditionedMRDTSig` *(*`AWSetF3`*, the LCA-blind lift of the binary inflation-separator, with a deflationary* rem*) satisfying every* `CoreVCs3` *and* `DeltaVCs3` *clause yet falsifying* `CDVC3`*, so* CoreVCs3 + DeltaVCs3 $\not\Rightarrow$ CDVC3 *(and, via* `joinLemma3_iff_cdVC3`*, $\not\Rightarrow$ the Join Lemma). The reason ternary closes where binary (b'') stays open is instructive:* `CDVC3`*'s $\sqsubseteq$-half demands update inflation ($A \sqsubseteq \mathsf{update}\, A\, e$), which the binary* `LatticeVCsPlus` *ships as a separate axiom but the ternary* `DeltaVCs3` *honestly omits — so the inflation-free delta laws cannot back* `CDVC3`*, and a deflationary update refutes it. The eight-VC bundle is therefore* ***minimal at*** `CDVC3`*: the causal-delta rule is a genuine independent axiom, not a derived lemma (a load-bearing point for the rule-basis of Open Question 15.1)*

Open Question 15.3 (structure theorem)**.** *Empirically, the unconditional delta contract holds exactly on the group and lattice classes. Is every unconditional-*DeltaVCs3 *merge a torsor-like group$\otimes$lattice combination? (The naïve two-sorted umbrella* $\mathsf{mergeL}(l, a, b) = a \boxplus (b \boxminus l)$ *does not derive the contract uniformly, so the question is genuinely open.)*

Open Question 15.4 (route reunification — naive route refuted, closure-indexed resolution mechanized)**.** *The obvious reunification — restate the delta laws and* CDVC3 *over full causal closure and re-run the Join Lemma induction — fails, and the failure is mechanized. The induction must peel an event that is both* lo^U*-maximal (placeable last) and vis-maximal (its removal keeps the sides fully closed); on a reachable four-event configuration these two maximal sets are disjoint, and the obstruction persists for every block peel, in both directions (*`JoinLemma3C`*:* `no_proper_back_block`*,* `no_proper_front_block`*). A wider induction class (fully-closed minus a* lo*-upward peel set) moves the obstruction but does not remove it: individually fully-closed*

sides admit no common witness set to decompose against (`killTest_no_common_U`). What does work needs no reunification: a ***closure-indexed contract*** JoinLemma3C $\mathcal{C}$ *in which the data type declares its closure strength* $\mathcal{C}$*, with a single soundness bridge (`ra_linearizable3_of_joinC`): store versions are fully causally closed, so* JoinLemma3C $\mathcal{C}$ *applies for any* $\mathcal{C}$ *that full closure implies — both weak and full qualify, and the whole `GoodConfig3` induction is reused verbatim. The two routes of §8 are the two instances, and all nine production discharges route through the bridge unchanged (`AllMRDTs_viaContract`), reusing their existing VCs verbatim: eight at* $(\mathit{weak}, \top)$ *— the OR-sets via the feasible triple, the six unconditional types via the delta on-ramp — and the enable-wins flag at* $(\mathit{full}, \top)$*; the disjunction is the contract, not a defect to be reunified. This is the "no new mathematics" the resolution promised — confirmed, with one correction: the bridge holds exactly for* $\mathcal{C}$ *weaker than full closure (a* $\mathcal{C}$ *stronger than full makes* JoinLemma3C *too weak to be adequate). Genuinely open: whether the single-strength contract (carrying the common witness set as data, re-founding* CDVC3 *on the full downset) buys anything over the closure-indexed form.*

Resolved (how the framework was found). An earlier open question — how to condition the update-layer VCs so that a reachability-conditioned data type like the tombstone-free RGA can be hosted — is settled: it *became* the conditioned metatheory of §10–§14. The refutation ladder that found it (the naive Inv-restriction refuted, `G2_Transport_Probe`; the applicability-aware order paired with no-op-feasible enumerations, `UpdateFeasibility_Gate`; the rehoming obstruction, `RGA_Rehoming_Gate`) is recorded in the mechanization map's investigation rows, and the resolution is the end-to-end theorem `rga_tombstone_free_ra_linearizable3_eq`. It is kept here as a pointer, not as an open question.

Open Question 15.5 (the automation layer)**.** *The published catalogue's seventeen induction-scheme VCs were an SMT-facing Skolemization of feasibility, aimed at the wrong target. Design the analogous per-data-type induction scheme whose induction implies* CDVC3 *and the feasible delta laws — restoring push-button discharge, now of VCs that actually imply RA-linearizability. The observed uniformity of the discharges (*σ*-characterization by sandwich invariant + absorber trichotomy) suggests the scheme exists.*

Open Question 15.6 (criss-cross merges and virtual LCAs)**.** *The transition system gates Merge on the existence of an LCA (§4.1); criss-cross configurations are reachable, and every version in them linearizes, but their heads cannot merge — the model is silent exactly where git reaches for its recursive strategy (merge the maximal common ancestors into a virtual base, then merge with that). The metatheory looks ready for the extension: the ternary Join Lemma is stated over backward-closed event sets, and* $E_1 \cap E_2$ *needs no witnessing store version. What remains is a virtual-LCA Merge rule plus one per-data-type question: do the eight VCs imply that the recursively computed base equals* $\sigma(E_1 \cap E_2)$*? For the counter this is inclusion–exclusion once more — exact iff the maximal common ancestors jointly cover the intersection, itself a candidate store invariant; for the OR-set, a pointwise Boolean check. A positive answer closes the one visible gap between the model and practice.*

Open Question 15.7 (rc-expressiveness: chains, LWW, and merge-recoverability)**.** *The catalogue's* rc *relations are all star-shaped: the conflicting pairs are cross-class (*rem*/*add*; add/checkout), so every edge runs from a class that is never second to a class that is never first. This is forced: the update layer's directionality law (`rc_non_comm_directional`) orients every non-commuting concurrent pair, while its companion `no_rc_chain` forbids length-2 paths, and any orientation of*

a self-conflicting class (concurrent writes; concurrent claims of one slot) contains one. So total-order policies — last-writer-wins first among them — are inexpressible in the rc *mechanism as constrained by the VC set. Two questions, one of them already closed at its boundary. (i) Would a chain-tolerant generalization (the concurrent arm of* lo *as a transitive tiebreak) be sound? The semantics is coherent — with Lamport clocks the composite of* vis *and any timestamp- or priority-oriented tiebreak is acyclic, and canonical states remain well-defined — so the restriction appears to be an artifact of the swap-flavored proof machinery (`cond_comm_lift` cannot bubble past two consecutive* rc *edges), not of the theorem. The decisive experiment: re-prove the swap VC and the causal-delta peel over a transitive tiebreak; we expect the swap route to break and the canonical-state route (which peels a global maximum, and a total order has one everywhere) to survive. (ii) What would it buy? Not a metadata-free LWW register: whatever the order arbitrates, the converged outcome must be produced by* $\mathsf{mergeL}(l, a, b)$ *— a function of three states — and for a plain-value LWW two executions present identical state triples with opposite timestamp orders, so no such function exists (`lww_merge_needs_timestamps`, `Refutations/LWW_Merge_Needs_Timestamps.lean`, kernel-checked). The arbitration key is forced into the state by the merge; once there, writes commute and* rc *is moot — which is why every LWW implementation stores its timestamp. What a chain-tolerant arm could buy: metadata-free priority policies across three or more op classes (outcomes state-recoverable, orientations chained), and possibly a lighter witness layer for the tombstone-free RGA — an* $\mathsf{ins} \prec \mathsf{del}$ *tiebreak keeps anchors live at fold time, which is exactly what the raw-fold witness discipline (§14) compensates for; any such attempt must reckon with the two standing refutations (dependency non-transitivity, `RGA_DepComp_Gate`; and the merge-side path requirement, which chains cannot remove — `RGA_PrefixFree_Impossible` is itself a merge-recoverability argument). The general principle the kill-test isolates:* rc *is the mechanism for conflicts whose outcome is recoverable from branch states; clique conflicts pay for their arbitration key in state, and chains would extend* rc *only within the recoverable class.*

Open Question 15.8 (Causal canonicity under general rc)**.** *The generic safety metatheorem consumes causal canonical witnesses — enumerations linearizing* $\xrightarrow{\mathsf{vis}}$ *and respecting* lo^E *jointly. With every concurrent pair unordered (Either), the two discharge species above suffice. Under a general* rc *the existence of a joint linearization is open: a cycle would need at least two* rc*-edges separated by* $\xrightarrow{\mathsf{vis}}$*-paths — a single* rc*-edge closed by a* $\xrightarrow{\mathsf{vis}}$*-path contradicts concurrency via transitivity, and a non-commuting first* $\xrightarrow{\mathsf{vis}}$*-step is an absorber killing the edge — and `no_rc_chain` forbids adjacent* rc*-edges but not* $\xrightarrow{\mathsf{vis}}$*-separated ones. `CausalCanonical` is therefore a hypothesis with two discharge lemmas rather than a theorem. The budget cart (`BudgetCart`, `MRDT_Instances/BudgetCart/`) has now forced the question — and widened it: its convergence discharges along the OR-set route, but the ungated safety obligation `SafetyStepOn` is false for it, by a two-event refutation — `CausalFold` pins only* $\xrightarrow{\mathsf{vis}}$*-respect, and the fold of a causal past containing a concurrent same-item* add/rem *pair is enumeration-dependent (the orders disagree exactly where add-wins should arbitrate). So for* rc*-nontrivial datatypes the honesty fold needs* rc*-orientation too: the right notion is folds along jointly* $\xrightarrow{\mathsf{vis}}$*- and* lo^E*-respecting enumerations, for both the version witnesses (`CausalCanonical`) and the causal-past folds (`CausalFold`). The cart's safety theorem is delivered gated (`bcart_version_inv_gated`, with the missing transfer named explicitly as `BCartSpendMono`); closing the gate is this question.*

Open Question 15.9 (Does conditioning compose? — convergence: yes, now a theorem)**.** *The catalogue is built one data type at a time; practice builds data types from data types — the canoni-*

cal case is the key-value map whose values are themselves MRDTs (the certified-MRDT line's own composite). The factored framework suggests composition should happen at the `JoinLemma3At` *boundary — the map combinator would consume each value's per-configuration join, whatever species produced it, so a map of queues composes as readily as a map of counters — with a small once-only kit: folds and configurations project per key, cross-key pairs commute so* lo^E *localizes, and per-key joins re-interleave. Conjecturally the coupling strength stratifies what survives composition: payload parametricity (free), the indexed product = the grow-only-keys map (a theorem to prove), read coupling — one layer's* app *reads another's state, as in a budgeted cart — (convergence composes; safety needs a monotone-transfer lemma), uniform coupling — key removal, reaching into the value layer the same way for every value type (one generic obligation: a reset element and a* remove $\|$ *v-op policy) — and ad-hoc effect coupling (rich-text marks anchored in a sequence), where nothing is expected to compose automatically. The convergence half is now mechanized (*`Metatheory/Product.lean`*): the binary heterogeneous product* $D_1 \otimes D_2$ *composes at exactly the* `JoinLemma3At` *boundary — cross-component pairs commute definitionally, so* lo^E *has no mixed edges and the glued join witness is a plain concatenation* $\iota_1\rho^1 +\!\!+ \iota_2\rho^2$ *(*`joinLemma3At_prod`*; composite metatheorem* `prod_ra_linearizable3_of_honest_reach`*, axioms* `propext`*,* `Quot.sound`*; consumability demo: a queue* $\otimes$ *counter capstone with zero bespoke proof,* `MRDT_Instances/ProductDemo/`*). Two boundary findings from the pen-and-paper pass (*`Development/COMPOSITION_PENPAPER.md`*): reachability does not project (a second-component apply is a version-duplicating stutter for the first's projection), so finished capstones never compose — only per-configuration certificates do, which retroactively forces this note's factored interface; and two-sided causal-witness re-interleaving is refuted (a realizable four-event cross-*$\xrightarrow{\mathsf{vis}}$ *cycle), with the sound repair — pin one side, re-sort the other — covering products with at most one* rc*-nontrivial component. The safety and* $\approx$ *kits are also mechanized: honesty,* `SafetyStep`*, and one-sided causal witnesses compose (*`Metatheory/Product_Safety.lean`*: the pinned-extension lemma,* `safetyStepOn_prod`*,* `causalCanonical_prod_of_one_sided`*, composite* `prod_version_inv_on_of_one_sided`*), and the eq-quotient layer lifts at the pragmatic cut* $\approx_2 = =$ *(*`Metatheory/ProductEq.lean`*: every* `GenericEqQuotient` *obligation componentwise,* `eqJoinLemma3C_H_prod` *glued by concatenation with no congruence chasing, capstone* `prod_ra_linearizable_up_to_eq_H` *— one quotiented component, one flat, exactly the Peritext shape, with the quotiented side's reachability supplies as premises per the memo's cost-center analysis). Still open: the general* $\approx_1 \times \approx_2$ *lift (no new ideas, double the plumbing — deferred until a second quotiented component exists), the indexed (map-of-MRDTs) generalization, and the L2.5/L3 coupling boundaries, where composition breaking is expected to be as informative as where it holds.*

Open Question 15.10 (The self-referential-spec limit, and document-order mark intent)**.** *RA-linearisability certifies that a version's state is the fold of some delivery-respecting linearisation of its events — convergence to the datatype's own* do*-fold, with that fold as the reference. It says nothing about whether the fold's single-replica semantics is the intended one; a defect in* do *appears identically in the concurrent state and its fold, and is certified as agreement. Independent intent specifications are therefore a separate obligation (the read-side theorems, the paper's examples), not a corollary of convergence.*

The cleanest witness of this needs no concurrency and no marks — three sequential operations on the tombstone-free RGA. Event 1 *inserts* a *after the root, event* 2 *inserts* b *after the root, and event* 3 *inserts* c *after* a*. The same-anchor siblings* a, b *read newest-first (*b *before* a*) and* c *sits under* a*, so the document reads* $[b, a, c]$*. Now delete* a*. Correct list semantics — and a*

tombstoned RGA — give $[b, c]$; the tombstone-free RGA gives $[c, b]$. Physically splicing a out rehomes its child c to the root, where c (event 3) out-ranks b (event 2) under the same newest-first tie-break and jumps ahead of it; a tombstoned RGA keeps a as a dead position-holder, so c never migrates. The reorder is baked into do *itself, so the merged state and the fold of every delivery-respecting linearisation agree on the same wrong sequence $[c, b]$, and RA-linearisability certifies that agreement without complaint — kernel-clean and RA-linearisable does not make the single-replica sequence the intended one. Only an independent order spec exposes the defect, and the fail/pass pair is machine-checked: the tombstone-free RGA refutes a "delete preserves survivor order" property (`tombstone_free_violates_delete_order` : ¬ `DeleteOrderPreserving`), via exactly this $[b, a, c] \to [c, b]$ execution, while the tombstoned RGA's order relation — invariant under a delete that touches only the tombstone set, not the insert records the read consults — satisfies it (`remove_preserves_visible_lt`). This is a more minimal instance of the limit than the Peritext leak below: three sequential operations, no concurrency, no marks.*

The original concrete instance was compositional: the tombstone-free Peritext composite's frozen-path mark resolution climbs tree ancestry and so leaks formatting backward under anchor deletion (§15); a paper-faithful account needs a document-order rehoming read (nearest surviving neighbour in reading order, gravity-aware), against which that leak is a visible failure. The fused boundary-node design provides such rehoming via the RGA's own machinery, at the cost of atomicity (the mark-positioning trilemma: atomic, tombstone-free, live — pick two), and it is now ***built*** *(`MRDT_Instances/Peritext/`): rich text as one RGA over* char ⊕ boundary*, with a document-order read and the genuine positional intent theorem `render_id_active_iff_between` — a rendered character carries a mark iff it lies between the mark's boundary nodes in reading order, i.e. fold-computed activation ⟺ a structural positional decomposition of the element list (not a restatement of the read, so it clears the circularity above). Its corollary `render_span_before` forbids the backward leak by construction — exactly the failure the frozen-path design retracts. So the concrete Peritext instance of this question is resolved for the fused design. What remains: (i) the enduring methodological point — convergence to the own-fold never certifies intent, only an independent spec does; and (ii) the RGA reorder above is, unlike the Peritext leak, not retractable — physical splicing is the data type, so the survivor reorder is intrinsic to tombstone-freedom rather than a bug to fix. It is thus the sharpest form of the point: a kernel-clean, RA-linearisable data type whose certified single-replica sequence is nevertheless not the sequential-list sequence, the gap visible only through an order property the metatheory never states.*

16 Mechanization

Everything above is mechanized in Lean 4 (Mathlib; axioms: `propext`, `Classical.choice`, `Quot.sound`; no `sorryAx` anywhere) in the Sal repository, under `Sal/ConditionedMRDTs/`:

Artifact	Lean	File
signature, ternary lo	`MRDTSig`, `ConditionedMRDTSig`	`Framework/MRDTSig.lean`
store, DAG, transitions	`Configuration`, `Step3`	`Framework/ExecutionModel.lean`, `Metatheory/LCA_Lemma.lean`
Lemma 4.1	`lca_events_of_storeInv`	`Metatheory/LCA_Lemma.lean`
σ/lo^E layer (merge-independent)	`UpdateVCs`, `IsCanonicalState`	`Framework/Sigma_LoOn3.lean`
the eight VCs	`CoreVCs3CD`, `FeasibleDeltaVCs3`, `CDVC3`	`Framework/VC_Set.lean`
Theorem 8.1 + bridges	`ra_linearizable_of_core_feasible_cd3`	`Metatheory/Adequacy.lean`
Table 1	all instance theorems	`MRDT_Instances/` (one directory per RDT)
bounded-counter safety	`bc_version_inv`, `bc_value_nonneg`	`MRDT_Instances/BoundedCounter/`
mergeable queue (direct join)	`q_join_at`, `queue_ra_linearizable3`	`MRDT_Instances/MergeableQueue/`
FWW register (payload arbitration)	`fww_version_min`	`MRDT_Instances/FWWRegister/`
BudgetCart (gated safety)	`BCart_ra_linearizable3_eq`, `bcart_version_inv_gated`	`MRDT_Instances/BudgetCart/`
per-configuration join hook	`JoinLemma3At`, `goodConfig3_merge_at`	`Framework/VC_Set.lean`, `Metatheory/Adequacy.lean`
honest-reachability metatheorem	`HonestReach`, `ra_linearizable3_of_honest_reach`	`Metatheory/HonestReach.lean`
generic honesty shape	`GenHonest`, `AppHonest`	`Metatheory/GenHonest.lean`
generic safety (causal witness)	`SafetyStep`, `version_inv_of_causal_canonical`	`Metatheory/GenericSafety.lean`
escrow metatheorem	`Measured`, `escrow_version_inv`	`Metatheory/EscrowSafety.lean`
product composition (raw kit)	`joinLemma3At_prod`, `prod_ra_linearizable3_of_honest_reach`	`Metatheory/Product.lean`
product safety kit	`safetyStepOn_prod`, `prod_version_inv_on_of_one_sided`	`Metatheory/Product_Safety.lean`
product $\approx$-lift	`eqJoinLemma3C_H_prod`, `prod_ra_linearizable_up_to_eq_H`	`Metatheory/ProductEq.lean`
Peritext by composition	`peritextComposed_ra_linearizable_up_to_eq`, `peritextRender_congr`	`MRDT_Instances/Peritext_Composed/`
Peritext, fused (positional intent)	`peritext_ra_linearizable_up_to_eq`, `render_id_active_iff_between`	`MRDT_Instances/Peritext/`
mark-anchor honesty composes	`MarkAccurate`, `peritextTF_ra_linearizable_honest`	`MRDT_Instances/Peritext_Composed/MarkHonesty.lean`
LWW merge needs timestamps (OQ 15.7)	`lww_merge_needs_timestamps`	`Refutations/LWW_Merge_Needs_Timestamps.lean`
CDVC3 independent of core+delta (OQ 15.2)	`cdvc3_not_derivable_from_core_delta`	`Refutations/CD_Not_Derivable_Ternary.lean`
impossibility kill-tests	—	`Refutations/Impossibility.lean`
findings journal T0–T11	—	`Development/MRDT_METATHEORY_DRAFT.md`
Conditioned-metatheory investigation (§4.2, Open Questions 15.4 & feasible-update; resolution in §10–§14):		
defeater vs. 2-OP (§4.2)	`no_inter_lca_2op_rem_peel_of_defeater`	`Refutations/InterLca2op_Defeater_Arbiter.lean`
peel obstruction (OQ 15.4)	`reunification_peel_obstruction`, `no_proper_back_block`	`Refutations/Reunification_Peel_Obstruction.lean`, `Metatheory/JoinLemma3C.lean`
closure-indexed contract	`JoinLemma3C`, `killTest_no_common_U`	`Refutations/JoinLemma3F_Of_AlmostClosed.lean`
closure-indexed adequacy + instances	`ra_linearizable3_of_joinC`, `ConditionedContract`	`Metatheory/ConditionedContract.lean`
all 9 discharges via the contract	`*_adequate_viaContract` (9)	`Development/AllMRDTs_viaContract.lean`
update-layer feasibility notion	`loOnA_noopFeasible_verdict`	`Refutations/UpdateFeasibility_Gate.lean`
conditioned convergence (order-half)	`conditioned_convergence_on`	`Framework/ConditionedConvergence.lean`
RGA rehoming: oracle false, converges	`verdict_oracle_false_but_converges`	`Refutations/RGA_Rehoming_Gate.lean`
feasible-update gate	`G2_conditioned_convergence_refuted`, `separating_inequivalence`	`Refutations/G2_{Transport_Probe,Applicability_Aware}.lean`
The conditioned metatheory (tombstone-free RGA end-to-end, feasible-update question resolved):		
merge-union witness refutation	—	`Development/RGA_HEnum_Refutation.lean`
observational quotient $D \mapsto D_{\approx}$	`QSig`, `qapplicable`	`Metatheory/GenericEqQuotient*.lean`
canonical-witness layer, raw-$\approx$ target	`IsRALinearizable3Eq`, `RA_linearizable_up_to_eq_H`	`Metatheory/GoodConfig3H.lean`
RGA canonical engine	`CanonInv`, `canon_fold`	`MRDT_Instances/RGA_TombstoneFree/RGA_Canon*.lean`
generation discipline from born accuracy	`genDisc2C_of_born`	`MRDT_Instances/RGA_TombstoneFree/RGA_GenDisc_Assembly.lean`
capstone + discharged leaves	`rga_RA_linearizable_skeleton3`	`MRDT_Instances/RGA_TombstoneFree/RGA_Skeleton3.lean`, `RGA_*_Discharge.lean`
honest residual	`HonestDelivery`, `rga_RA_linearizable_honest`	`MRDT_Instances/RGA_TombstoneFree/RGA_Honest_Residual.lean`
mainline entry point	`rga_tombstone_free_ra_linearizable3_eq`	`MRDT_Instances/RGA_TombstoneFree/RA_Lin.lean`
flat collapse of the framework	`eqJoinH_of_joinC`, `flat_ra_linearizable3_eq`	`Metatheory/FlatGeneric_Bridge.lean`
the flat instances, generically	`*_ra_linearizable3_eq` (11)	`MRDT_Instances/` (per-RDT)

The update-side layer (lo^E, convergence, σ) is merge-independent — it is stated over the update signature $\langle \Sigma, \sigma_0, \mathsf{do}, \mathsf{rc} \rangle$ alone — and is shared with the binary (CRDT) specialization of the theory; in the repository it lives in `Sal/CRDTs/Metatheory/`, the directory that also holds the binary exactness results quoted in Open Question 15.2. The production data types of Table 1 are mirrored faithfully from `Sal/MRDTs/*` (encoding deviations documented per instance); their original 24-VC discharges are untouched.

The foundations live under `Framework/`, the metatheorems under `Metatheory/`, the machine-checked negative results under `Refutations/`, and the per-RDT conditioned instances under `MRDT_Instances/` (the tombstone-free RGA chain in its `RGA_TombstoneFree/` directory); findings journals and superseded routes remain under `Development/`, which nothing imports. Each cited theorem is kernel-checked with axioms in {`propext`, `Classical.choice`, `Quot.sound`} (no `sorryAx`). They import one linearization file carrying two unrelated `sorry`s that the cited theorems do not transitively depend on; `Development/CONDITIONED_METATHEORY_PLAN.md` is their roadmap.