Exact Float Counter CRDTs: Lawful, Machine-Checked Convergence for Replicated Floating-Point Sums

Kyle Clouthier
Clouthier Simulation Labs
kyle@simgen.dev https://simgen.dev

July 12, 2026

Abstract

Counter CRDTs (Conflict-free Replicated Data Types) have been *integer-valued* for fifteen years. The reason is algebraic: a state-based counter's merge must be commutative, associative, and idempotent, and IEEE-754 floating-point addition is neither commutative nor associative *in the accumulated state*—so a floating-point sum cannot lawfully instantiate the standard construction, and its replicas do not converge to identical values. We observe that an *exact, mergeable superaccumulator*—a fixed-point integer wide enough to hold every finite `f64` exactly—restores precisely the missing properties, making the standard state-based counter-CRDT construction lawful for floating-point sums. We give the construction, machine-check its convergence laws in Lean 4 (the merge is a join-semilattice; folding *any* delivery schedule—arbitrary reordering and duplication—yields the same state; merges are monotone; and the converged read equals the exactly rounded sum of every increment), and provide a canonical byte encoding so replicas prove convergence by comparing a hash rather than an ε. We are deliberate about novelty. The superaccumulator is classical [5, 6, 7]; the counter-CRDT template is standard [3]; the contribution is the *lawful instantiation for exact floating-point*, the mechanized convergence laws, and a torture-tested open-source implementation. We situate the result against mechanized-CRDT work [4], which verifies an *integer* counter within a network model; our added dimension is exactness for continuous values, not a new verification framework. Code, proofs, and a reproducible torture test are public.

1 Introduction

A replicated counter that any node can increment, that tolerates arbitrary message reordering, duplication, and partition, and that converges without coordination, is one of the foundational Conflict-free Replicated Data Types [3]. The grow-only counter (G-Counter) and its signed cousin (PN-Counter) are textbook. Every widely deployed implementation we are aware of counts *integers*: Redis Enterprise Active-Active documents 59-bit integer counters [9]; Akka's `GCounter` is `BigInt`-valued; Riak's counters are integers.

This is not an oversight. A state-based CRDT converges because its merge $\sqcup$ forms a *join-semilattice*: commutative, associative, and idempotent [3]. A G-Counter meets this by storing one non-negative integer per replica and merging entrywise with max, which inherits those properties from the integers. If one tries to replace the per-replica integer with a floating-point running sum, the construction breaks at the algebra: IEEE-754 addition is non-associative—$(a \oplus b) \oplus c \neq a \oplus (b \oplus c)$ in general—so two replicas that receive the same increments in different orders hold different bit

patterns, and merging those states is neither well-defined nor convergent. A recent generic counter library makes the requirement explicit by being parametric over a commutative semigroup; `f64` under $\oplus$ is simply not a lawful argument. The result is a fifteen-year gap: no lawful floating-point counter CRDT.

The gap is worth closing because floating-point aggregates that must converge are common: replicated metrics, offline-first sums that reconcile on sync, multi-region billing totals that different replicas compute in different orders. The usual workaround is to ban floats from replicated state entirely—the deterministic-simulation-testing discipline does exactly this—or to accept "eventually approximately equal" and reconcile with a tolerance, which forfeits the CRDT's exact, hash-checkable convergence.

Contribution. We make one observation and follow it through with proofs and code.

1. **Observation.** An *exact, mergeable* accumulator—a wide fixed-point integer (a *superaccumulator*) that represents every finite `f64` exactly—makes floating-point summation associative and commutative *in the accumulated state, by construction*, because integer addition is. This restores exactly the algebra the counter-CRDT construction requires.
2. **Construction.** The standard state-based counter recipe, instantiated with this accumulator, yields a G-Counter whose value is an *exactly rounded* floating-point sum (§3).
3. **Mechanized convergence.** We model the construction in Lean 4 and machine-check its convergence laws: the merge is a join-semilattice; folding any delivery schedule (any permutation, any duplicates) yields the same state; merges are monotone; and the converged full-knowledge read equals the exact sum of every increment (§4). The proofs carry no `sorry` and use only Lean's standard axiom base.
4. **Hash-checkable convergence.** The accumulator has a canonical byte encoding, so converged replicas prove agreement by comparing a hash rather than an ε (§5).
5. **Implementation.** An open-source Rust implementation, torture-tested across randomized gossip schedules with duplicate and stale delivery, converges byte-identically where a naive `f64` counter does not (§6).

What this is not. We claim no new algorithm and no new verification framework. The superaccumulator is classical [5, 6, 7]; the counter-CRDT template is standard [3]; machine-checked CRDTs exist [4]. The novelty is the composition—a *lawful, exact* floating-point counter CRDT with mechanized convergence laws and a canonical encoding—and the honest engineering that makes it checkable. We state prior art up front (§7) and invite correction: if an exact floating-point counter CRDT with mechanized laws already exists, we have not found it.

2 Background

State-based CRDTs. A state-based (convergent) CRDT is a join-semilattice $(S, \sqcup)$: replicas hold states in S, update locally by moving up the lattice, and merge with $\sqcup$, which must be commutative, associative, and idempotent. Strong eventual consistency follows: replicas that have observed the same set of updates hold equal state, regardless of message order, duplication, or delay [3].

The G-Counter. A grow-only counter over replicas $\{0, \ldots, R-1\}$ is a map $g : \mathrm{Fin}\, R \to \mathbb{N}$. Replica r increments by raising its own entry $g(r)$; $(g \sqcup h)(r) = \max(g(r), h(r))$; the value is $\sum_r g(r)$. Commutativity, associativity, and idempotence of $\sqcup$ are inherited from max on $\mathbb{N}$. Only the owner writes its entry, so concurrent increments never conflict.

Why floats break it. Replace $g(r) \in \mathbb{N}$ with a floating-point running sum $s(r) \in \texttt{f64}$ accumulated from replica r's increment stream. Because $\oplus$ is non-associative, $s(r)$ depends on the *order* in which r applied its increments, and two replicas relaying r's increments in different orders disagree on $s(r)$ bit-for-bit. "Highest count wins" has nothing to select on, and merging by re-adding floats compounds the non-associativity. The read $\sum_r s(r)$ is itself order-dependent. Convergence to identical values fails.

Exact superaccumulators. An exact summation accumulator represents the running sum as a wide fixed-point integer rather than a float. Every finite `f64` is an integer multiple of 2^{-1074}; a two's-complement integer of a few thousand bits holds any such value, and any sum of a bounded number of them, *exactly*. Integer addition into this accumulator is associative and commutative, so the accumulated state is independent of summation order by construction; a single correctly-rounded conversion at read time recovers an `f64`. The idea is classical: Kulisch's exact dot-product accumulator [5], Neal's superaccumulators [6], and the Demmel–Nguyen reproducible-summation line [7]. Our implementation uses a 2176-bit two's-complement accumulator in units of 2^{-1074}; the details of the accumulator are not the contribution and we treat it as a black box with one property, stated next.

Definition 1 (Exact mergeable accumulator)**.** *An* exact mergeable accumulator *is a type A with an injection $\iota : \mathit{f64}_{\mathrm{fin}} \to \mathbb{Z}$ (the exact value in units of 2^{-1074}), an empty state $\bot_A$ with model value 0, an operation* $\mathsf{add} : A \times \mathit{f64}_{\mathrm{fin}} \to A$ *whose model is $s \mapsto s + \iota(x)$, and a merge* $\mathsf{merge} : A \times A \to A$ *whose model is integer addition. It exposes a monotone* $\mathsf{count} : A \to \mathbb{N}$ *(the number of* add*s) and a canonical encoding* $\mathsf{bytes} : A \to \{0,1\}^*$ *with* $\mathsf{bytes}(a) = \mathsf{bytes}(b) \iff a = b$.

Because the accumulator's state is an integer under addition, merge is exactly commutative and associative—the two properties floating-point addition lacks and the counter-CRDT construction requires.

3 The Exact Float Counter CRDT

We instantiate the standard state-based G-Counter with the exact accumulator of Definition 1.

State. A replica holds a map from replica id to an exact accumulator, $\mathsf{GC} : \mathrm{Fin}\, R \to A$, with entry r owned by replica r.

Increment. To add a floating-point value x, replica r applies $\mathsf{add}(-, x)$ to its own entry. Entries are append-only: a replica only ever adds to its own accumulator, so its sequence of own-states is totally ordered by count.

Merge. The join is entrywise "highest count wins":

$$(s \sqcup t)(r) \;=\; \begin{cases} s(r) & \text{if } \mathsf{count}(s(r)) \geq \mathsf{count}(t(r)), \\ t(r) & \text{otherwise.} \end{cases}$$

Because entry r is append-only and owned by r, the entry with the larger count strictly contains the smaller one's increments; equal counts imply equal state. This is the standard counter-CRDT join, and it is where exactness matters: two relayed copies of replica r's accumulator at the same count are *byte-identical* precisely because the accumulator is order-invariant—the property a float sum lacks.

Read. The counter's value is $\mathsf{value}(\bigsqcup_r \mathsf{merge\text{-}all})$: merge every entry's accumulator and convert once, with a single correct rounding. Merge order is irrelevant by the accumulator's associativity/-commutativity.

Idempotence and deduplication. As with every counter CRDT, the *entry-level* join is idempotent (re-merging an entry at the same count is a no-op), but merging an accumulator's raw state into itself would double-count; deduplication is the map layer's responsibility, exactly as in the integer case [3]. What the exact accumulator supplies is the missing piece for floats: a deterministic, exact, commutative-associative entry state with a canonical encoding.

4 Machine-Checked Convergence

We model the construction in Lean 4 (core only, no `mathlib`) and prove its convergence laws. The model reflects the append-only invariant: because entry r is append-only and owned by r, an entry is faithfully represented by *how many* increments it contains, so "highest count wins" on same-replica entries is max on those counts (equal counts imply equal state, by construction). A counter state is one count per replica, $\mathsf{GC}\, R = \mathrm{Fin}\, R \to \mathbb{N}$, and the join is the pointwise max. The exact-value facts (that folding increments in any order gives the same integer state, and that the read is the exactly rounded sum) are inherited from a separate, independently machine-checked order-invariance result for the accumulator itself.

Theorem 2 (Join semilattice)**.** *The join is commutative, associative, and idempotent:* $s \sqcup t = t \sqcup s$, $(s \sqcup t) \sqcup u = s \sqcup (t \sqcup u)$, *and* $s \sqcup s = s$.

Theorem 3 (Delivery-schedule invariance)**.** *Let* joinAll *fold a list of received snapshots into a replica's state. For any two lists related by a permutation,* joinAll *yields the same state; and receiving the same snapshot twice in succession changes nothing. Consequently* any *delivery schedule—arbitrary reordering and arbitrary duplicate deliveries—produces the same state.*

Theorem 4 (Monotonicity)**.** *For all* s, t *and* r, $s(r) \le (s \sqcup t)(r)$*: a merge never discards an increment either side knows.*

Theorem 5 (Exact read)**.** *The full-knowledge state—every increment of every replica—reads the exact sum of every increment. Combined with the accumulator's order-invariance, this is byte-for-byte the state any ordering or sharding of those same increments produces.*

The Lean development discharges Theorems 2–5 (as `join_comm`, `join_assoc`, `join_idem`, `joinAll_perm_invariant`, `joinAll_dup_invariant`, `le_join_left`, and `read_full_exact`), plus an absorption lemma showing the full-knowledge state is the top of the reachable lattice. A CI job audits every theorem's axiom dependencies: all reduce to Lean's standard base (`propext`, `Classical.choice`, `Quot.sound`); none uses `sorryAx`. The order-invariance of the accumulator (that folding increments in any order, or by any merge tree, gives the same integer state) and the correctness of the final rounding are proved in a companion development and additionally checked at the bit level by bounded model checking of the Rust implementation.

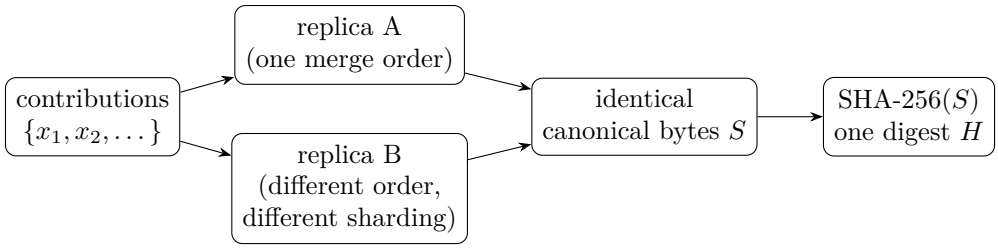

Figure 1: Hash-checkable convergence. Two replicas that receive the same contributions in different merge orders and shardings compute *byte-identical* canonical state S, hence the same 32-byte digest H; agreement is a single hash comparison, not an ε judgment. An ordinary floating-point sum has no such canonical state to hash.

Honest scope of the proofs. The Lean proofs establish the value-level algebra of the construction. That the Rust map layer increments `count` per `add` and applies "highest-count-wins" per entry is a small trusted surface, exercised (not proved) by the randomized torture test of §6. Deduplication semantics are the map layer's, as in every counter CRDT. We claim mechanized *convergence laws for the construction*, not end-to-end verification of a production system.

5 Canonical Encoding and Hash-Checkable Convergence

The exact accumulator serializes to a fixed-size canonical byte string with $\mathsf{bytes}(a) = \mathsf{bytes}(b) \iff a = b$ (Definition 1). A counter state therefore has a canonical encoding (the concatenation of its entries' encodings under a fixed replica order), and two replicas that have converged hold byte-identical state. This upgrades convergence testing from an approximate, threshold-based comparison to an exact one: replicas prove agreement by exchanging and comparing a hash of the canonical bytes, and any disagreement is a single-bit event rather than an ε judgment call. For float aggregates this is qualitatively new—an ordinary float sum has no canonical state to hash—and it is what makes the CRDT's convergence *auditable* rather than merely asymptotic.

The implementation exposes this as a first-class primitive (the `receipts` feature, v0.2): `state_hash` returns the 32-byte SHA-256 of the canonical encoding, so two parties that hold the same multiset—in any order, any sharding, any merge tree—obtain the same digest, and any dropped, duplicated, or altered contribution changes it. This makes an aggregate *proof-carrying*: signing is deliberately out of scope (bring your own keys—ed25519, ML-DSA, whatever the trust plane uses), and a verifier recomputes the aggregate from the shards it trusts and compares the signed digest. The security reduction is clean: agreement on a floating-point statistic reduces to agreement on a hash, and forging a different multiset with the same aggregate reduces to a hash collision rather than to hiding inside an ε tolerance. Because the digest commits to the exact state (not an order-dependent float), the same commitment is reproducible on heterogeneous hardware—the property that makes a signed aggregate meaningful across replicas that never shared an execution order.

6 Implementation and Evaluation

We implement the construction in Rust over an exact 2176-bit superaccumulator with a 289-byte canonical encoding. The construction is exercised by a randomized adversarial harness intended to break it:

- eight replicas, each adding its own stream of hostile values (mixed magnitudes across many decades, subnormals, and exact-cancellation pairs);
- 300 randomized gossip schedules, each interleaving local increments with snapshot exchanges that include *duplicate delivery* and *stale snapshots*, followed by a partition-heal anti-entropy sweep;
- after quiescence, every replica's canonical state and every converged value are compared.

Across all 300 schedules every replica converged *byte-identically*, and every converged value equalled the exactly rounded sum of all increments. As a contrast, re-summing the *same* converged per-replica values in forward versus reverse order with naive `f64` addition produced different bit patterns in 184 of 300 schedules—evidence that exactness is load-bearing here, not decorative. The same code compiles to WebAssembly and reproduces the identical canonical hash as native x86-64 and ARM64, so cross-architecture replicas agree bit-for-bit.

We report cost honestly. Exact accumulation is roughly 5–10× a naive summation loop and 1.2–2.5× compensated (Kahan) summation on mixed-magnitude data; merging shard states is effectively free (hundreds of nanoseconds per merge). This is the price of an exact, mergeable, canonically-encoded state, and it is paid where bit-identity matters—replicated aggregates, signed/hashed outputs, cross-machine reconciliation—not in a hot inner loop.

7 Related Work

CRDTs and counters. Convergent and commutative replicated data types, and the join-semilattice characterization of state-based convergence, are due to Shapiro et al. [3]. Counter CRDTs in that framework, and in every deployment we know of, are integer-valued: Redis Enterprise Active-Active documents 59-bit integer counters [9]; Akka and Riak counters are integers. Generic counter libraries are parametric over a commutative semigroup, which `f64` under $\oplus$ does not lawfully inhabit—the precise gap this note closes.

Mechanized CRDTs. Gomes, Kleppmann, Mulligan, and Beresford give an Isabelle/HOL framework with an explicit network model and machine-checked strong eventual consistency for an increment/decrement counter, an Observed-Remove set, and a Replicated Growable Array [4]. Their framework is more thorough on the distributed-systems side than our development—it models the network and proves consistency over all network behaviours—and their counter is the *integer* case. Our contribution is orthogonal: exactness for a *floating-point* aggregate, via a lawful accumulator, with the convergence laws mechanized. We do not contribute a verification framework and we do not model the network; combining an exact-accumulator counter with a network-model framework such as theirs is natural future work.

Exact and reproducible summation. The exact accumulator we rely on is classical: Kulisch's exact accumulator [5], Neal's superaccumulators [6], and the Demmel–Nguyen / ReproBLAS reproducible-summation line [7]. These make summation order-independent on one machine; we use one such accumulator as the lawful monoid a counter CRDT needs. Reproducible aggregation has also been studied in databases [8], single-node and without a mergeable or serializable accumulator state.

The honest question. The individual ingredients are known. We searched for an exact floating-point counter CRDT with mechanized convergence laws and did not find one; if it exists, we would like to cite it and correct this note.

8 Beyond counters: convergent statistics

The construction generalizes past counters. Any statistic whose sufficient state is a set of *exact monomial sums* ($\sum x_i$, $\sum x_i^2$, $\sum x_i^3$, $\sum x_i^4$, $\sum x_i y_i$, …) inherits the whole contract: each sum lives in its own exact accumulator, so the composite state is a product of lawful monoids and the merge laws lift componentwise (and pointwise for keyed families); these lifting laws are machine-checked in `proofs/ToolkitAlgebra.lean`. Reads are computed from the exact integer state in big-integer arithmetic with a *single* final round-to-nearest-even, so every read that is a rational function of the state is the *correctly rounded value of the true statistic*: mean, population and sample variance, covariance, least-squares slope and intercept, and R^2. Two cases are worth noting: Pearson kurtosis μ_4/μ_2^2 and squared skewness μ_3^2/μ_2^3 are exactly rounded because the sample-size and unit factors cancel, leaving a pure rational of the state; reads involving a square root (standard deviation, correlation, skewness) apply one IEEE-correctly-rounded sqrt to an exactly rounded radicand—at most two roundings, still bit-identical everywhere. Exact retraction (a positive/negative pair of moment states) gives insert-then-delete reads that return to byte-identical values, and exact covariance *matrices* support multiple linear regression whose normal-equation entries are exactly rounded (the solve itself is a fixed deterministic elimination: bit-invariant, not exactly rounded—a stated second tier).

The classical mergeable-moments art makes the contrast precise. The pairwise update formulas of Chan, Golub, and LeVeque [1] are *algebraically* exact but computed in floating point: the resulting bits depend on the merge tree, and the merge double-counts on re-delivery. Mergeable summaries [2] are mergeable by design but approximate. Reproducible reductions in the ReproBLAS lineage achieve bit-identity by *pinning* an operation order. Exact, order-invariant-without-pinning, and CRDT-lawful appears to be an unoccupied combination; as with the counter itself, we state this as a question rather than a claim of priority. The generalization ships in the implementation's `stats` feature (v0.2), with each read verified against an independent big-integer oracle by a neighbor-comparison correct-rounding check, including catastrophic-cancellation regimes (mean 10^8, spread 10^{-3}: the textbook one-pass formula returns a *negative* variance; the reads here are exactly rounded). The two-product domain limits apply per monomial and are detected, never silent.

9 Limitations

The Lean proofs establish the construction's value-level algebra, not an end-to-end verification of the implementation; the map layer's per-entry count and dedup are trusted and tested rather than proved (§4). Like every counter CRDT, deduplication of repeated shards is the map layer's job; merge alone is not idempotent. The accumulator's cost (§6) makes this suitable for aggregates where bit-identity is the point, not for high-throughput inner loops. Finally, the statistics of §8 inherit these limits per monomial sum (narrowing the certified input domain for third and fourth moments), and order statistics (medians, quantiles) admit no exact mergeable state at all—fixed-bucket histograms with exact counts and honest quantile *bounds* are the boundary of the family.

10 Conclusion

Counter CRDTs were integer-only because their merge must be commutative and associative and floating-point addition, in the accumulated state, is neither. An exact, mergeable superaccumulator restores exactly those properties, making the standard construction lawful for floating-point sums. The result is a floating-point counter CRDT whose convergence laws are machine-checked, whose converged value is an exactly rounded sum, and whose agreement is provable by a hash rather than an ε. The primitives are old and cited; the contribution is the lawful composition, the mechanized laws, and a public, torture-tested implementation. We hope the honest framing—prior art first, a named question about missing work, and calibrated claims—is itself useful in a subfield where the interesting move is often a lawful instantiation rather than a new algorithm.

Artifacts. The implementation, the Lean proofs (with the axiom-audit CI job), and the randomized torture test are available as open source (MIT/Apache-2.0) at `https://github.com/KyleClouthier/bitrep`; the exact float counter CRDT is the `float_gcounter` example, the convergence proofs are `proofs/FloatGCounter.lean`, and the convergent statistics of §8 are the `stats` feature (v0.2 on crates.io) with merge-algebra proofs in `proofs/ToolkitAlgebra.lean`. A browser demonstration of the underlying exact accumulator is at `https://simgen.dev/bitrep/`.

References

[1] T. F. Chan, G. H. Golub, and R. J. LeVeque. *Algorithms for Computing the Sample Variance: Analysis and Recommendations.* The American Statistician 37(3), 1983.

[2] P. K. Agarwal, G. Cormode, Z. Huang, J. M. Phillips, Z. Wei, and K. Yi. *Mergeable Summaries.* PODS 2012.

[3] M. Shapiro, N. Preguiça, C. Baquero, and M. Zawirski. *Conflict-free Replicated Data Types.* SSS 2011.

[4] V. B. F. Gomes, M. Kleppmann, D. P. Mulligan, and A. R. Beresford. *Verifying Strong Eventual Consistency in Distributed Systems.* OOPSLA 2017 / PACMPL 1(OOPSLA).

[5] U. Kulisch. *Computer Arithmetic and Validity: Theory, Implementation, and Applications.* de Gruyter, 2008.

[6] R. M. Neal. *Fast Exact Summation Using Small and Large Superaccumulators.* arXiv:1505.05571, 2015.

[7] J. Demmel and H. D. Nguyen. *Fast Reproducible Floating-Point Summation.* IEEE ARITH 2013; see also ReproBLAS.

[8] I. Müller, A. Arteaga, T. Hoefler, and G. Alonso. *Reproducible Floating-Point Aggregation in RDBMSs.* ICDE 2018.

[9] Redis. *Active-Active database strings and counters (59-bit integer counters).* Redis Enterprise documentation.