MDPI

Article

Deterministic Replay Verification of Interval Programs over a Finite Primitive Core via Quantifier-Free Integer Certificates

Hidemitsu Maeki

[1] GhostDrift Mathematical Institute (Kabushiki Kaisha GhostDrift Suri Kenkyujo), 4-4-4 Kita-Shinjuku, Shinjuku-ku, Tokyo 169-0074, Japan; manny@ghostdriftresearch.com

Abstract

The Analytically Derived Interval Computation Integrity Certificates (ADIC) framework is a replay-verification framework for interval-based computations over a fixed finite primitive core. The paper does not introduce a new family of interval operators. Instead, it shows that the verifier obligations for this primitive core can be reduced to finite, instantiated, quantifier-free integer checks over a realized ledger. The framework combines a fixed-point encoding of real intervals, a strict Galois insertion, explicit Euclidean witnesses for nontrivial primitive rules, and a replayable compilation of program and specification constraints. For the fixed primitive core considered here, verifier acceptance is proved to imply the existence of a concrete trajectory and enclosure of the certified specification constraints. The verifier performs deterministic replay over the realized program and specification ledgers, with linear dependence on ledger size up to integer bit-complexity. The claim concerns the verifier core only; transfer to deployed floating-point implementations is handled separately through an explicit implementation-inclusion assumption.

Keywords: interval verification; formal verification; fixed-point abstraction; Galois insertion; quantifier-free certificate checking; proof-carrying verification

1. Introduction

As autonomous systems and cyber–physical devices become ubiquitous, their numeric components—such as control algorithms and neural networks—must satisfy stringent safety requirements. Independent verification of these bounds is fundamentally complicated by the non-determinism of hardware floating-point implementations [1–6] and the intractable computational complexity of non-linear real arithmetic constraint solving.

To address this, we package numeric safety arguments into finite artefacts that can be checked deterministically with strictly bounded computational resources [7]. The scope of this article is replay verification for interval-based computations generated over the fixed primitive set $\Sigma_{\mathrm{prim}} = \{+, -, \times, \mathrm{inv}, \mathrm{sqrt}, \mathrm{relu}\}$. The contribution is not a new family of interval operators, nor a general complexity result for nonlinear-real verification. Rather, we prove that, for this fixed primitive core, the verifier can be closed inside a quantifier-free ground integer signature Σ_{int}: all verification obligations are normalized into instantiated integer checks over a finite realized ledger, and each nontrivial primitive rule is discharged by explicit Euclidean witnesses. Transfer to deployed floating-point behavior is handled separately by an explicit implementation-inclusion contract.

This paper makes the following theorem-level contributions for the fixed primitive core:

Received:
Revised:
Accepted:
Published:

1. A formalization of fixed-point interval abstraction as a Galois insertion obtained from integer well-ordering [12].
2. Closed-form integer rules for the primitive set $\Sigma_{\text{prim}} = \{+, -, \times, \text{inv}, \text{sqrt}, \text{relu}\}$, where each nontrivial primitive soundness obligation is discharged by explicit Euclidean witnesses.
3. A total normalization map τ from certificate-side checks to a minimal quantifier-free ground integer signature, so that every formula evaluated by the verifier is an instantiated integer formula under standard integer semantics.
4. An end-to-end theorem for the fixed primitive core showing that verifier acceptance implies the existence of a concrete trajectory, enclosure of the certified specification constraints, and deterministic replay verification with cost $O((n+s) \cdot C(b))$ for a realized certificate ledger. This is a replay bound for a realized certificate, not a general complexity claim for nonlinear-real solving.

As public companion artifacts, we also provide (i) a Lean formalization of a core lemma in ADIC as a machine-checkable proof artifact, and (ii) an ADIC-based electricity-demand audit demonstration with published certificates and an audit report for a real January–April 2024 electricity-demand $\times$ weather case study. These companion artifacts support the surrounding formal and deployment-oriented context of the framework, but they are not part of the theorem-level claims proved in the present paper.

Analytically Derived Interval Computation Integrity Certificates (ADIC) verification pipeline (conceptual).

Program directed acyclic graph (DAG) + witnesses $\rightarrow$ program-ledger replay over pure integers $\rightarrow$ specification directed acyclic graph (Spec-DAG) compilation $\rightarrow$ specification-ledger replay $\rightarrow$ ACCEPT/REJECT with deterministic $O((n+s)\,C(b))$ checks.

Figure 1. Overview of the certificate and replay-verification workflow.

2. Related Work and Positioning

Interval analysis and standards.

Interval arithmetic is a mature discipline with standardization (IEEE 1788) and extensive foundational and survey treatments [8–10]. Recent work has also pushed interval arithmetic directly into mechanized program-verification settings [11]. These works primarily define sound interval operations and their mathematical intent, but they do not, by themselves, prove that a verifier for a fixed primitive core can be closed inside a minimal, ground integer signature. In contrast, the theorem-level novelty of the present paper is not a new operator semantics result. It is a verifier-closure result for the fixed primitive set Σ_{prim}: via a total normalization τ, all verifier obligations are reduced to instantiated ground Σ_{int} checks, and each nontrivial primitive rule is discharged by explicit Euclidean witnesses.

Proof-carrying code and replay-style verification.

Proof-carrying code (PCC) established the paradigm of shipping machine-checkable evidence with programs [7]. Recent certified-checker work likewise emphasizes independent checking of proof artefacts and reducing trust in the checker itself [13]. The Analytically Derived Interval Computation Integrity Certificates (ADIC) framework follows this philosophy but specializes it to numeric safety of interval-based computations over the fixed primitive core: the certificate is a finite ledger of instantiated, quantifier-free integer inequalities and Euclidean witnesses, and verification is a deterministic *replay* procedure over that ledger. The key positioning difference is that the proof object is narrowed to instantiated integer safety obligations and the checker performs only replay of ground Σ_{int} instances, with no search over real arithmetic.

Abstract interpretation and Galois connections.

Abstract interpretation provides the general theory for sound over-approximation by abstract domains [12]. While our construction is compatible with this viewpoint, the contribution here is narrower and more concrete: for the fixed primitive core, we prove a strict Galois insertion between real interval semantics and an encoded fixed-point integer domain, and we close each primitive soundness obligation by explicitly witnessed integer inequalities. Thus, the novelty is not a general abstract-interpretation framework, but closure of the concrete verifier obligations themselves as finite ground checks.

Floating-point proof tools and SMT baselines.

Floating-point non-determinism and rounding-aware reasoning are addressed by tools such as Gappa and FPTaylor, and by more recent systems such as PRECiSA 4.0, VCFloat2, and end-to-end Coq-based floating-point verification [1–6]. We do not incorporate such tool reasoning into the verifier trusted computing base (TCB); instead, deployment behavior is separated as an explicit implementation-inclusion assumption, while the verifier remains purely integer-based. As a baseline, solver-oriented encodings of analogous constraints in non-linear real arithmetic can become computationally heavy [14,15]. Our theorem is narrower: for the fixed primitive core and a realized certificate ledger, solver-driven search is replaced by deterministic replay in $O((n+s)\,C(b))$. This is not a general complexity claim for nonlinear-real verification.

Table 1. Positioning of Analytically Derived Interval Computation Integrity Certificates (ADIC) against neighboring lines of work.

Line of work	Primary object	Checking-time burden	Distinguishing point of the framework
Interval standards and classical interval analysis	Sound interval operators and outer bounds	Mathematical semantics of interval operations; not a closed replay verifier by itself.	Closes verifier obligations for a fixed primitive core inside ground integer replay.
Proof-carrying code	Machine-checkable evidence attached to programs	Proof object depends on the target logic and checker.	Specializes proof-carrying checking to a finite ledger of instantiated integer inequalities and Euclidean witnesses.
Floating-point proof tools	Rounding-aware reasoning for floating-point expressions	Checker depends on floating-point or analytic reasoning.	Separates deployment semantics by explicit contract while keeping the verifier core discrete.
SMT / QF_NRA encodings	Solver-based constraint satisfaction over nonlinear reals	Search-heavy solving can dominate checking cost.	Replaces solver search by deterministic replay over a finite realized ledger for the fixed primitive core.

3. Materials and Methods

This article is a theoretical mathematics paper centered on verifier design, proof closure, and complexity analysis for a fixed finite primitive core of interval programs. The workflow is: (i) define the fixed-point interval domain and outward abstraction; (ii) restrict the verifier to a ground quantifier-free integer signature via a total normalization homomorphism; (iii) close all primitive rules for Σ_{prim} by explicit Euclidean witnesses; (iv) serialize both

program evaluation and specification checking into finite ledgers; and (v) replay-verify a realized certificate in deterministic linear time. The mathematical constructions and verifier obligations are formalized in Sections 5–18.

4. Results

For the fixed primitive core $\Sigma_{\text{prim}} = \{+, -, \times, \text{inv}, \text{sqrt}, \text{relu}\}$, the main theorem-level conclusion of this paper is the following: for a realized certificate ledger, verifier acceptance implies the existence of a concrete trajectory, enclosure of all certified specification evaluations, and deterministic replay verification in time $O((n+s)\cdot C(b))$. Table 2 summarizes the formal components that establish this claim.

Table 2. Main formal outputs established in this paper.

Formal output	Precise claim	Scope / theorem anchor
Strict abstraction result	Real intervals embed into the encoded fixed-point domain via a strict Galois insertion with optimal outward integer bounds.	Theorem 1
Ground verifier closure	Every verifier obligation is reduced to an instantiated quantifier-free integer formula with a unique truth value under standard integer semantics.	Lemma 7, Lemma 9
End-to-end soundness	ACCEPT implies existence of a concrete trajectory, enclosure of all compiled specification evaluations, and satisfaction of all certified inequalities.	Theorem 9, Theorem 10, Theorem 11
Deterministic replay complexity	Verification of a realized certificate is linear in realized program and specification ledger sizes, up to multi-precision integer bit-complexity.	Theorem 12

Internal worked certificate example.

To make the replay semantics concrete, we include a fully internal worked certificate example. At scale $D = 1$, take input intervals $I_1 = [1, 2]$ and $I_2 = [3, 5]$. The four endpoint products are $\{3, 5, 6, 10\}$, so replay of the explicit minimum and maximum witnesses yields the certified multiplication enclosure $[3, 10]$. For the specification $e(x) = x - 10$, specification replay gives the output enclosure $[-7, 0]$. Therefore, the verifier accepts the certified safety claim $e(x) \leq 0$. This example is illustrative rather than empirical: its role is to expose the finite certificate semantics and to show that the verifier consumes instantiated integer checks rather than performing search over real arithmetic.

Nature of the contribution.

Accordingly, this article is a theoretical mathematics paper with an internal worked certificate example rather than a benchmark study. Its scientific contribution is the mathematical closure of the verifier core and the replay-verification architecture for the fixed primitive core Σ_{prim}.

5. Formal Foundations of Pure-Integer Abstraction

We define the mathematical foundation required to translate continuous operations into purely integer-based verification obligations. Let $\mathbb{Z}$ denote the ring of integers, $\mathbb{N}_{>0}$

the strictly positive integers, and $\mathbb{R}$ the field of real numbers. We fix a global scaling factor $D \in \mathbb{N}_{>0}$.

5.1. Integer Extrema and Well-Ordering

To define the integer abstraction without appealing to limiting constructions, we construct the interval boundaries directly from the well-ordering principle of integers.

Lemma 1 (Existence of Integer Maximum). *Let $S \subset \mathbb{Z}$ be a non-empty set bounded above by $M \in \mathbb{Z}$ such that $\forall s \in S, s \leq M$. Then S possesses a unique maximum element* $\max S \in S$.

Proof. Let $T := \{M - s \mid s \in S\}$. Since $s \leq M$, $T \subset \mathbb{N}_{\geq 0}$. Because S is non-empty, T is non-empty. By the well-ordering principle of $\mathbb{N}_{\geq 0}$, T contains a unique minimum $t_0 = M - s_0$. For any $s \in S$, $M - s \geq t_0 = M - s_0$, which implies $s \leq s_0$. Thus $s_0 = \max S$. □

Lemma 2 (Existence of Integer Minimum). *Let $S \subset \mathbb{Z}$ be a non-empty set bounded below by $m \in \mathbb{Z}$ such that $\forall s \in S, m \leq s$. Then S possesses a unique minimum element* $\min S \in S$.

Proof. Let $T := \{-s \mid s \in S\} \subset \mathbb{Z}$. T is bounded above by $-m$. By Lemma 1, $\max T$ exists. We define $\min S = -\max T$. □

5.2. Abstract Domain and Galois Insertion

Definition 1 (Encoded Interval Domain). *The abstract domain of encoded fixed-point intervals is defined as:*

$$\mathcal{I}_D := \{(a, b) \in \mathbb{Z}^2 \mid a \leq b\}$$

The concretization function $\gamma_D : \mathcal{I}_D \to 2^{\mathbb{R}}$ decodes an integer pair into a continuous real interval:

$$\gamma_D(a, b) := [a/D,\ b/D] \subset \mathbb{R}$$

We define the partial order $(a, b) \sqsubseteq (c, d) \iff (c \leq a) \wedge (b \leq d)$. This logically coincides with subset inclusion: $(a, b) \sqsubseteq (c, d) \iff \gamma_D(a, b) \subseteq \gamma_D(c, d)$.

Lemma 3 (Boundedness of Abstraction Sets). *For any $\ell, u \in \mathbb{R}$ with $\ell \leq u$ and $D \in \mathbb{N}_{>0}$, the integer sets $S_\ell := \{a \in \mathbb{Z} \mid a/D \leq \ell\}$ and $S_u := \{b \in \mathbb{Z} \mid u \leq b/D\}$ are non-empty, and respectively bounded above and below.*

Proof. By the Archimedean property of $\mathbb{R}$, there exists $n \in \mathbb{N}_{>0}$ such that $n > -D\ell$. Thus $-n/D < \ell$. Choosing $a = -n \in \mathbb{Z}$ yields $a/D \leq \ell$, proving S_ℓ is non-empty. Furthermore, there exists $m \in \mathbb{N}_{>0}$ such that $m > D\ell$, meaning $\ell < m/D$. For any $a \in S_\ell$, $a/D \leq \ell < m/D$, implying $a < m$. Since $a, m \in \mathbb{Z}$, $a \leq m - 1$. Thus S_ℓ is bounded above by $m - 1$. Symmetrically for S_u: there exists $m' \in \mathbb{N}_{>0}$ such that $m' > Du$, so $u < m'/D$. Choosing $b = m' \in \mathbb{Z}$ gives $u \leq b/D$, meaning S_u is non-empty. There exists $n' \in \mathbb{N}_{>0}$ such that $n' > -Du$, so $u > -n'/D$. For any $b \in S_u$, $-n'/D < u \leq b/D$, so $-n' < b$, meaning $-n' + 1 \leq b$. Thus S_u is bounded below. □

Definition 2 (Outward Abstraction via Integer Extrema). *Let $\mathcal{J} := \{[\ell, u] \subset \mathbb{R} \mid \ell \leq u\}$ be the set of bounded, closed real intervals. For $J = [\ell, u] \in \mathcal{J}$, we construct the optimal outer integer boundaries purely via integer selection. By Lemmas 1, 2, and 3, we uniquely define:*

$$A(J) := \max S_\ell, \qquad B(J) := \min S_u$$

Since $\ell \leq u$, it strictly follows that $A(J)/D \leq \ell \leq u \leq B(J)/D$, ensuring $A(J) \leq B(J)$. We define the pure-integer abstraction function as $\alpha_D(J) := (A(J), B(J)) \in \mathcal{I}_D$.

Lemma 4 (Equivalence to Floor and Ceiling). *The optimal boundaries strictly correspond to mathematical floor and ceiling operations:* $A([\ell, u]) = \lfloor D\ell \rfloor$ *and* $B([\ell, u]) = \lceil Du \rceil$.

Proof. $A(J) = \max\{a \in \mathbb{Z} \mid a/D \leq \ell\} = \max\{a \in \mathbb{Z} \mid a \leq D\ell\}$. By definition of the floor function over reals, the largest integer not exceeding $D\ell$ is exactly $\lfloor D\ell \rfloor$. Symmetrically, $B(J) = \min\{b \in \mathbb{Z} \mid Du \leq b\} = \lceil Du \rceil$. □

Theorem 1 (Strict Galois Insertion). *The functions* (α_D, γ_D) *form a Galois insertion between* $(\mathcal{J}, \subseteq)$ *and* $(\mathcal{I}_D, \sqsubseteq)$*, ensuring strict monotonicity and continuous bounding. Specifically:*

1. $J \subseteq \gamma_D(\alpha_D(J))$
2. $\alpha_D(\gamma_D(I)) = I$
3. *(Adjunction)* $\alpha_D(J) \sqsubseteq I \iff J \subseteq \gamma_D(I)$

Proof. (1) Let $J = [\ell, u]$. By Definition 2, $A(J)/D \leq \ell$ and $u \leq B(J)/D$. Thus, $J = [\ell, u] \subseteq [A(J)/D, B(J)/D] = \gamma_D(\alpha_D(J))$. (2) Let $I = (c, d) \in \mathcal{I}_D$. Then $\gamma_D(I) = [c/D, d/D]$. $A(\gamma_D(I)) = \max\{x \in \mathbb{Z} \mid x/D \leq c/D\} = c$. Similarly, $B(\gamma_D(I)) = \min\{y \in \mathbb{Z} \mid d/D \leq y/D\} = d$. Thus $\alpha_D(\gamma_D(I)) = (c, d) = I$. (3) Let $J = [\ell, u] \in \mathcal{J}$ and $I = (c, d) \in \mathcal{I}_D$. Let $\alpha_D(J) = (A(J), B(J))$. By the definition of $\sqsubseteq$, $\alpha_D(J) \sqsubseteq I \iff (c \leq A(J)) \wedge (B(J) \leq d)$. We claim $c \leq A(J) \iff c/D \leq \ell$. Indeed, if $c \leq A(J)$ then $c/D \leq A(J)/D \leq \ell$. Conversely, if $c/D \leq \ell$, then $c \in S_\ell$. Because $A(J) = \max S_\ell$, $c \leq A(J)$. Symmetrically, $B(J) \leq d \iff u \leq d/D$. Therefore, $\alpha_D(J) \sqsubseteq I$ holds if and only if $(c/D \leq \ell) \wedge (u \leq d/D)$, which equivalently means $J = [\ell, u] \subseteq [c/D, d/D] = \gamma_D(I)$. □

5.3. Set Extensions and Bounded Hulls

Lemma 5 (Existence of Hull). *If* $S \subset \mathbb{R}$ *is non-empty and bounded, then* $\inf S, \sup S \in \mathbb{R}$ *exist by the completeness of* $\mathbb{R}$*, and* $\text{hull}(S) := [\inf S, \sup S] \in \mathcal{J}$.

Definition 3 (Set Extension). *For any function* $f : \mathbb{R}^k \to \mathbb{R}$*, its set extension is* $f^\sharp(S_1, \ldots, S_k) := \{f(x_1, \ldots, x_k) \mid x_i \in S_i\}$.

Lemma 6 (Monotonicity). *If* $S_i \subseteq S_i'$ *for all* i*, then* $f^\sharp(S_1, \ldots, S_k) \subseteq f^\sharp(S_1', \ldots, S_k')$*. For non-empty bounded sets,* $A \subseteq B \implies \text{hull}(A) \subseteq \text{hull}(B)$.

6. Signatures, Normalization, and Quantifier-Free Compilation

To completely close the formal verification system, we strictly isolate the algebraic signature from arbitrary logical connectives.

Definition 4 (Verification Signature Σ_{int}, Source Signature Σ_{src}, and Instantiated Formulas). *The verifier arithmetic logic evaluates exclusively ground (fully instantiated) quantifier-free formulas. The strict operator signature is:*

$$\Sigma_{int} := \{+, -, \times, \leq, <, =\}$$

We additionally define a source *signature that may occur in certificate-side checks:*

$$\Sigma_{src} := \Sigma_{int} \cup \{>, \geq, \neq, \implies\} \cup \{\text{true}\}.$$

We treat true *as an abbreviation for the ground atom* $(0 = 0)$*. Let* `IntTerm` *be terms composed solely of integer constants and* $\{+, -, \times\}$*. Let* `Atom` *be predicates of the form* $(t_1 = t_2)$*,* $(t_1 < t_2)$*,* $(t_1 \leq t_2)$*. A* `QFForm` *(Quantifier-Free Formula) is defined recursively from* `Atoms` *using* $\{\wedge, \vee, \neg\}$.

A check schema *is a* `QFForm` *over* Σ_{src} *written with formal integer parameters (placeholders). For a schema* $\phi(\xi_1, \ldots, \xi_m)$ *and integers* $(z_1, \ldots, z_m)$*, we write* $\phi[z_1, \ldots, z_m]$ *for the* instantiated

formula obtained by simultaneously replacing each placeholder ξ_j by z_j. The verifier only evaluates such instantiated formulas, hence all formulas it evaluates are ground.

Lemma 7 (Ground Determinism). *For any instantiated `QFForm` ϕ over Σ_{int}, the truth value of ϕ under standard integer semantics is uniquely determined.*

Proof. By recursion on the finite syntax tree of ϕ: each `IntTerm` evaluates to a unique integer in $\mathbb{Z}$, each `Atom` to a unique boolean, and the connectives $\wedge, \vee, \neg$ are truth-functional. □

Lemma 8 (Trichotomy of Integers). *For any $a, b \in \mathbb{Z}$, exactly one of the following propositions is true: $a = b$, $a < b$, or $b < a$. These states are mutually exclusive and exhaustive.*

Proof. For $a, b \in \mathbb{Z}$, consider $a - b \in \mathbb{Z}$. Exactly one of $a - b = 0$, $a - b > 0$, $a - b < 0$ holds by the total order of $\mathbb{Z}$. These correspond respectively to $a = b$, $b < a$, and $a < b$. Mutual exclusivity follows from antisymmetry and irreflexivity of $<$. □

Definition 5 (Total Normalization Homomorphism τ). *We define τ as a total recursive homomorphism mapping instantiated formulas over Σ_{src} into Σ_{int} atoms and connectives. For operations natively within the target space, τ is the identity mapping. It preserves boolean structure: $\tau(\neg\phi) := \neg\tau(\phi)$, $\tau(\phi \wedge \psi) := \tau(\phi) \wedge \tau(\psi)$, $\tau(\phi \vee \psi) := \tau(\phi) \vee \tau(\psi)$. We additionally set $\tau(\text{true}) := (0 = 0)$. Derived relational symbols are strictly replaced:*

1. $\tau(A > B) := (B < A)$
2. $\tau(A \geq B) := (B \leq A)$
3. $\tau(A \neq B) := (A < B) \vee (B < A)$
4. $\tau(P \implies Q) := \neg\tau(P) \vee \tau(Q)$

Lemma 9 (Truth Preservation of τ). *For any instantiated formula ϕ over Σ_{src}, ϕ evaluates to true under standard integer semantics if and only if $\tau(\phi)$ evaluates to true.*

Proof. By structural induction on the grammar of instantiated formulas. The base atoms are identical. The case $\phi = \text{true}$ reduces to $(0 = 0)$ by definition of $\tau(\text{true})$. The expansions of $>$, $\geq$, and $\implies$ hold via fundamental propositional and algebraic equivalences. The expansion of $\neq$ is logically equivalent by the strict Trichotomy of integers (Lemma 8). Connectives are preserved directly. □

6.1. Explicit Witness Division Inequalities

We trace Euclidean division properties directly via certificate-supplied witnesses. We explicitly reject unbounded loops, `case` statements, and unverified floor/ceiling implementations from the evaluation engine.

Definition 6 (Abstract Division Bounds $\text{Check}_{\text{FDIV}}$, $\text{Check}_{\text{CDIV}}$). *For $z \in \mathbb{Z}$ and $d \in \mathbb{Z} \setminus \{0\}$, given a target bound $Q \in \mathbb{Z}$, define:*

$$\begin{aligned}\text{Check}_{FDIV}(z, d, Q) := {} & (0 < d \wedge Q \cdot d \leq z \wedge z < (Q+1) \cdot d) \\ & \vee (d < 0 \wedge Q \cdot d \geq z \wedge z > (Q+1) \cdot d) \\ \text{Check}_{CDIV}(z, d, Q) := {} & (0 < d \wedge (Q-1) \cdot d < z \wedge z \leq Q \cdot d) \\ & \vee (d < 0 \wedge (Q-1) \cdot d > z \wedge z \geq Q \cdot d).\end{aligned}$$

https://doi.org/10.3390/math1010000

Lemma 10 (Negative Divisor Inequality Rule). *For any integers A, B and $d < 0$, dividing inequalities by d reverses the relation over $\mathbb{R}$:*

$$A \leq B \implies A/d \geq B/d, \qquad A < B \implies A/d > B/d.$$

Proof. Assume $d < 0$ and view A, B, d as real numbers. Since $-1/d > 0$, multiplying $A \leq B$ by $-1/d$ preserves the inequality:

$$(-1/d)A \leq (-1/d)B.$$

But $(-1/d)A = -(A/d)$ and $(-1/d)B = -(B/d)$, hence

$$-(A/d) \leq -(B/d).$$

Multiplying both sides by -1 reverses the inequality, giving $A/d \geq B/d$. The strict case $A < B$ is identical. □

Lemma 11 (FDIV Bounding). *If $\text{Check}_{FDIV}(z, d, Q)$ is true, the mathematical ratio is strictly bounded as $Q \leq z/d < Q + 1$.*

Proof. If the $(0 < d)$ branch is true: $Qd \leq z < (Q + 1)d$. Dividing all terms by $d > 0$ preserves inequalities, yielding $Q \leq z/d < Q + 1$. If the $(d < 0)$ branch is true: $Qd \geq z > (Q + 1)d$. Applying Lemma 10, dividing by $d < 0$ reverses the inequalities, identically yielding $Q \leq z/d < Q + 1$. □

Lemma 12 (CDIV Bounding). *If $\text{Check}_{CDIV}(z, d, Q)$ is true, the mathematical ratio is strictly bounded as $Q - 1 < z/d \leq Q$.*

Proof. If $(0 < d)$: $(Q - 1)d < z \leq Qd$. Dividing by $d > 0$ preserves inequalities: $Q - 1 < z/d \leq Q$. If $(d < 0)$: $(Q - 1)d > z \geq Qd$. Applying Lemma 10, dividing by $d < 0$ reverses inequalities: $Q - 1 < z/d \leq Q$. □

Lemma 13 (Unified Division Bounding). *Let $z \in \mathbb{Z}$ and $d \in \mathbb{Z} \setminus \{0\}$. If $\text{Check}_{\text{FDIV}}(z, d, Q)$ holds then $Q \leq z/d < Q + 1$. If $\text{Check}_{\text{CDIV}}(z, d, Q)$ holds then $Q - 1 < z/d \leq Q$.*

Proof. Both statements are exactly Lemma 11 and Lemma 12 with the two sign branches. The negative-divisor branch correctness is discharged solely by Lemma 10. □

Lemma 14 (Reciprocal Monotonicity on Sign-Stable Domains). *Let $x, y \in \mathbb{R}$ with $xy > 0$ (i.e., both are positive or both are negative). If $x < y$ then $1/x > 1/y$. Equivalently, the function $r(x) = 1/x$ is strictly decreasing on $\mathbb{R}_{>0}$ and on $\mathbb{R}_{<0}$.*

Proof. Assume $xy > 0$ and $x < y$. Then

$$\frac{1}{x} - \frac{1}{y} = \frac{y - x}{xy}.$$

Here $y - x > 0$ and $xy > 0$, so $(y - x)/(xy) > 0$, hence $1/x - 1/y > 0$, i.e., $1/x > 1/y$. □

7. The Finite Ruleset Σ_{prim} and Direct Soundness

We fix the primitive ruleset $\Sigma_{\text{prim}} = \{+, -, \times, \text{inv}, \text{sqrt}, \text{relu}\}$. Division is not taken as a primitive; at compilation time it is represented as $x/y := x \times \text{inv}(y)$.

Definition 7 (Implementation Semantics and Domain Bound). *For each $\sigma \in \Sigma_{prim}$, let $f_\sigma^{impl} : \mathbb{R}^{k(\sigma)} \to \mathbb{R}$ denote the mathematical semantics of the primitive on the domain $\mathrm{Dom}_\sigma^{\mathbb{R}} \subseteq \mathbb{R}^{k(\sigma)}$. We pair this with an abstract indicator $\mathrm{Dom}_\sigma : \mathcal{I}_D^{k(\sigma)} \to$ `QFForm`.*

To guarantee mathematical safety prior to invocation, we explicitly define and prove the Abstract Domain Safety bounds for every primitive.

Lemma 15 (Abstract Domain Safety for inv). *Let $\mathrm{Dom}_{inv}(a,b) := (0 < a) \vee (b < 0)$. If $\tau(\mathrm{Dom}_{inv}(a,b))$ is true, then $\gamma_D(a,b) \subseteq \mathbb{R} \setminus \{0\}$.*

Proof. By Lemma 9, the ground proposition $(0 < a) \vee (b < 0)$ is true. If $0 < a$, then for any $x \in \gamma_D(a,b)$, $x \geq a/D > 0$. Thus $\gamma_D(a,b) \subseteq \mathbb{R}_{>0}$. If $b < 0$, then for any $x \in \gamma_D(a,b)$, $x \leq b/D < 0$. Thus $\gamma_D(a,b) \subseteq \mathbb{R}_{<0}$. In both disjoint continuous regions, $0 \notin \gamma_D(a,b)$. □

Lemma 16 (Abstract Domain Safety for sqrt). *Let $\mathrm{Dom}_{sqrt}(a,b) := (0 \leq a)$. If $\tau(\mathrm{Dom}_{sqrt}(a,b))$ is true, then $\gamma_D(a,b) \subseteq \mathbb{R}_{\geq 0}$.*

Proof. By Lemma 9, $0 \leq a$. For any $x \in \gamma_D(a,b)$, $x \geq a/D \geq 0$. Thus $\gamma_D(a,b) \subseteq \mathbb{R}_{\geq 0}$. □

Lemma 17 (Abstract Domain Safety for Trivial Domains). *For $\sigma \in \{+, -, \times, \mathrm{relu}\}$, we set $\mathrm{Dom}_\sigma :=$ true. Because their mathematical domains are $\mathbb{R}^k$, the relation $\gamma_D(\dots) \subseteq \mathrm{Dom}_\sigma^{\mathbb{R}}$ is unconditionally satisfied.*

Definition 8 (Well-Formedness WF* and R-SOUND). *For each σ, the Verifier strictly evaluates the composite ground formula:*

$$\mathrm{Check}_\sigma^*(I_1, \dots, I_k, W_\sigma) := \mathrm{Dom}_\sigma(I_1, \dots, I_k) \wedge \mathrm{Check}_\sigma(I_1, \dots, I_k, W_\sigma)$$

*A node evaluation is structurally well-formed, denoted WF_σ^*, if and only if $\tau(\mathrm{Check}_\sigma^*)$ evaluates to true. A closed-form algorithm $T_\sigma(I_1, \dots, I_k, W_\sigma)$ returning $(A, B) \in \mathbb{Z}^2$ satisfies **R-SOUND** if, whenever WF_σ^* holds:*

1. *$A \leq B$, ensuring $(A, B) \in \mathcal{I}_D$ (Closure).*
2. *$(f_\sigma^{impl})^\sharp(\gamma_D(I_1), \dots, \gamma_D(I_k)) \subseteq \gamma_D(A, B)$ (Soundness).*

We construct R-SOUND entirely via explicit Euclidean integer traces. Let $I = (a, b)$ and $J = (c, d) \in \mathcal{I}_D$, implying $a \leq b$ and $c \leq d$.

7.1. Addition and Subtraction

$f_+^{\mathrm{impl}}(x, y) = x + y$, $f_-^{\mathrm{impl}}(x, y) = x - y$. $\mathrm{Check}_+, \mathrm{Check}_- :=$ true.
Closed Form: $T_+(I, J) := (a + c,\ b + d)$ and $T_-(I, J) := (a - d,\ b - c)$.

Theorem 2 (R-SOUND for +, -). *T_+ and T_- satisfy R-SOUND.*

Proof. Closure is trivially $a \leq b \wedge c \leq d \implies a + c \leq b + d$. For Soundness of addition: take any $x \in [a/D, b/D]$ and $y \in [c/D, d/D]$. Adding the inequalities yields $a/D + c/D \leq x + y \leq b/D + d/D$. Factoring $1/D$ proves $x + y \in [(a + c)/D, (b + d)/D] = \gamma_D(T_+)$. A symmetric substitution proves T_-. □

7.2. Multiplication

$f_\times^{\mathrm{impl}}(x, y) = xy$.

Lemma 18 (Bilinear Vertex Extremum). *Let $x_1 \le x_2$ and $y_1 \le y_2$ be real numbers and define $X = [x_1, x_2]$, $Y = [y_1, y_2]$. Then the extrema of $f(x, y) = xy$ on $X \times Y$ are attained at the four vertices:*

$$\min_{(x,y)\in X\times Y} xy = \min\{x_1y_1, x_1y_2, x_2y_1, x_2y_2\}, \qquad \max_{(x,y)\in X\times Y} xy = \max\{x_1y_1, x_1y_2, x_2y_1, x_2y_2\}.$$

Proof. Fix $y \in Y$. The map $x \mapsto xy$ is affine on X, hence its minimum and maximum on X are attained at $x \in \{x_1, x_2\}$. Define

$$g(y) := \min\{x_1y, x_2y\}, \qquad h(y) := \max\{x_1y, x_2y\}.$$

Then for every $(x, y) \in X \times Y$ we have $g(y) \le xy \le h(y)$, and both bounds are attainable by choosing $x = x_1$ or $x = x_2$.

Next, observe that g and h are piecewise affine in y (each is the pointwise min/max of two affine maps). On any interval where g is affine, its extrema over Y occur at the endpoints y_1 or y_2; the same holds for h. Therefore,

$$\min_{y\in Y} g(y) = \min\{g(y_1), g(y_2)\}, \qquad \max_{y\in Y} h(y) = \max\{h(y_1), h(y_2)\}.$$

But $g(y_1), g(y_2), h(y_1), h(y_2)$ are exactly the four products $x_1y_1, x_1y_2, x_2y_1, x_2y_2$ arranged by min/max, hence the claim follows. □

Lemma 19 (Fixed-Point Product Bounding). *Let $x \in [a/D, b/D]$ and $y \in [c/D, d/D]$ with $a \le b$ and $c \le d$. Then*

$$\frac{\min\{ac, ad, bc, bd\}}{D^2} \le xy \le \frac{\max\{ac, ad, bc, bd\}}{D^2}.$$

Proof. Apply Lemma 18 with $x_1 = a/D, x_2 = b/D, y_1 = c/D, y_2 = d/D$ and factor $1/D^2$ from each vertex product. □

Witness: $W_\times = (p_{\min}, p_{\max}, Q_f, Q_c) \in \mathbb{Z}^4$. Let $p_1 = ac, p_2 = ad, p_3 = bc, p_4 = bd$. To ensure compliance with the quantifier-free verifier language without relying on unverified min/max macros, the checks explicitly bind the minimum and maximum of this finite set:

$$\begin{aligned}\mathrm{Check}_{\min}(a, b, c, d, p_{\min}) := {} & (p_{\min} = p_1 \vee p_{\min} = p_2 \vee p_{\min} = p_3 \vee p_{\min} = p_4) \\ & \wedge (p_{\min} \le p_1) \wedge (p_{\min} \le p_2) \\ & \wedge (p_{\min} \le p_3) \wedge (p_{\min} \le p_4)\end{aligned}$$

$$\begin{aligned}\mathrm{Check}_{\max}(a, b, c, d, p_{\max}) := {} & (p_{\max} = p_1 \vee p_{\max} = p_2 \vee p_{\max} = p_3 \vee p_{\max} = p_4) \\ & \wedge (p_1 \le p_{\max}) \wedge (p_2 \le p_{\max}) \\ & \wedge (p_3 \le p_{\max}) \wedge (p_4 \le p_{\max})\end{aligned}$$

Lemma 20 (Min Witness Correctness). *If $\tau(\mathrm{Check}_{\min}(a, b, c, d, p_{\min}))$ is true, then*

$$p_{\min} = \min\{ac, ad, bc, bd\}.$$

Proof. By Lemma 9, $\mathrm{Check}_{\min}$ holds under standard integer semantics. The first conjunct forces $p_{\min} \in \{p_1, p_2, p_3, p_4\}$ where $p_1 = ac, p_2 = ad, p_3 = bc, p_4 = bd$. The remaining conjuncts state $p_{\min} \le p_i$ for each $i = 1, 2, 3, 4$. Therefore $p_{\min}$ is an element of the set that is $\le$ every element of the set, hence it is exactly the minimum. □

Lemma 21 (Max Witness Correctness). *If* $\tau(\text{Check}_{\max}(a,b,c,d,p_{\max}))$ *is true, then*

$$p_{\max} = \max\{ac, ad, bc, bd\}.$$

Proof. By Lemma 9, $\text{Check}_{\max}$ holds. The first conjunct forces $p_{\max} \in \{p_1, p_2, p_3, p_4\}$. The remaining conjuncts state $p_i \le p_{\max}$ for each $i = 1, 2, 3, 4$. Therefore $p_{\max}$ is an element of the set that is $\ge$ every element of the set, hence it is exactly the maximum. □

Lemma 22 (Min/Max Ordering). *If* $\tau(\text{Check}_{\min}(a,b,c,d,p_{\min}) \wedge \text{Check}_{\max}(a,b,c,d,p_{\max}))$ *is true, then* $p_{\min} \le p_{\max}$.

Proof. From Lemma 20 and Lemma 21, $p_{\min}$ is the minimum and $p_{\max}$ is the maximum of the same finite set, hence $p_{\min} \le p_{\max}$. □

Check Definition:

$$\begin{aligned}\text{Check}_{\times}(I, J, W_{\times}) := \ & \text{Check}_{\min}(a,b,c,d,p_{\min}) \\ & \wedge \text{Check}_{\max}(a,b,c,d,p_{\max}) \\ & \wedge \text{Check}_{\text{FDIV}}(p_{\min}, D, Q_f) \\ & \wedge \text{Check}_{\text{CDIV}}(p_{\max}, D, Q_c).\end{aligned}$$

We rely on the absolute constant $D > 0$.

Closed Form: $T_{\times}(I, J, W_{\times}) := (Q_f, Q_c)$.

Theorem 3 (R-SOUND for $\times$). *$T_{\times}$ satisfies R-SOUND.*

Proof. Closure: If $\text{WF}^*_{\times}$ passes, Lemma 22 guarantees $p_{\min} \le p_{\max}$. By Lemma 13 applied to divisor $D > 0$, we have $Q_f \le p_{\min}/D \le p_{\max}/D \le Q_c$. Thus $Q_f \le Q_c$, meaning $T_{\times} \in \mathcal{I}_D$. Soundness: For $x \in \gamma_D(I)$ and $y \in \gamma_D(J)$, we have $x \in [a/D, b/D]$ and $y \in [c/D, d/D]$. By Lemma 19,

$$\frac{\min\{ac, ad, bc, bd\}}{D^2} \le xy \le \frac{\max\{ac, ad, bc, bd\}}{D^2}.$$

By Lemma 20 and Lemma 21, $p_{\min} = \min\{ac, ad, bc, bd\}$ and $p_{\max} = \max\{ac, ad, bc, bd\}$. From Lemma 13 with divisor $D > 0$, $Q_f \le p_{\min}/D$ and $p_{\max}/D \le Q_c$. Dividing by $D > 0$ yields $Q_f/D \le p_{\min}/D^2$ and $p_{\max}/D^2 \le Q_c/D$. Therefore $Q_f/D \le xy \le Q_c/D$, i.e., $xy \in \gamma_D(T_{\times})$. □

7.3. Reciprocal

$f^{\text{impl}}_{\text{inv}}(x) = 1/x$.

Witness: $W_{\text{inv}} = (Q_L, Q_U) \in \mathbb{Z}^2$.

Check Definition: $\text{Check}_{\text{inv}}(a, b, Q_L, Q_U) := \text{Check}_{\text{FDIV}}(D \cdot D, b, Q_L) \wedge \text{Check}_{\text{CDIV}}(D \cdot D, a, Q_U)$.

Closed Form: $T_{\text{inv}}(a, b, W_{\text{inv}}) := (Q_L, Q_U)$.

Theorem 4 (R-SOUND for inv). *T_{inv} satisfies R-SOUND.*

Proof. By Abstract Domain Safety (Lemma 15), WF^*_{inv} guarantees $0 \notin [a/D, b/D]$, thus $1/x$ is totally defined. We evaluate the non-overlapping sub-domains. Closure: If $0 < a$, then $0 < a \le b$. By Lemma 13, $Q_L \le D^2/b$ and $D^2/a \le Q_U$. By Lemma 14, the reciprocal map is strictly decreasing on each sign-stable domain, hence for $0 < a \le b$ we have $1/b \le 1/a$. Thus $Q_L \le D^2/b \le D^2/a \le Q_U$. If $b < 0$, then $a \le b < 0$. By Lemma 13 applied to the

explicitly negative divisor $b < 0$, the ratio evaluation correctly applies Lemma 10 to yield $Q_L \leq D^2/b$. By Lemma 13 on divisor $a < 0$, $D^2/a \leq Q_U$. By Lemma 14, the reciprocal map is strictly decreasing on each sign-stable domain, hence for $a \leq b < 0$ we also have $1/b \leq 1/a$. Thus $Q_L \leq D^2/b \leq D^2/a \leq Q_U$. Soundness: For any $x \in [a/D, b/D]$. If $0 < a$, then $1/x \in [D/b, D/a]$. From Lemma 13 with positive divisor $b > 0$, $Q_L \leq D^2/b$, so $Q_L/D \leq D/b \leq 1/x$. From Lemma 13 with positive divisor $a > 0$, $D^2/a \leq Q_U$, so $1/x \leq D/a \leq Q_U/D$. If $b < 0$, then $1/x \in [D/b, D/a]$. From Lemma 13 with negative divisor $b < 0$, inequality is reversed strictly to $Q_L \leq D^2/b$, thus $Q_L/D \leq D/b \leq 1/x$. From Lemma 13 with negative divisor $a < 0$, inequality is reversed to $D^2/a \leq Q_U$, thus $1/x \leq D/a \leq Q_U/D$. Thus $1/x \in \gamma_D(T_{\text{inv}})$. □

7.4. Square Root

$$f^{\text{impl}}_{\text{sqrt}}(x) = \sqrt{x}.$$

Witness: $W_{\text{sqrt}} = (p, q) \in \mathbb{Z}^2$.
Check Definition: $\text{Check}_{\text{sqrt}}(a, b, p, q) := (0 \leq p) \wedge (0 \leq q) \wedge (p \cdot p \leq a \cdot D) \wedge (b \cdot D \leq q \cdot q) \wedge ((0 < b \cdot D) \vee (q = 0))$.
Closed Form: $T_{\text{sqrt}}(a, b, W_{\text{sqrt}}) := (p, q)$.

Theorem 5 (R-SOUND for sqrt). *T_{sqrt} satisfies R-SOUND.*

Proof. By Abstract Domain Safety (Lemma 16), $\text{WF}^*_{\text{sqrt}}$ guarantees $\gamma_D(I) \subseteq \mathbb{R}_{\geq 0}$, ensuring real square roots exist. Closure: $p^2 \leq aD \leq bD \leq q^2$. Since $p \geq 0$ and $q \geq 0$, $p \leq q$, ensuring $T_{\text{sqrt}} \in \mathcal{I}_D$. Soundness: For $x \in [a/D, b/D]$, multiplying by $D > 0$ gives $0 \leq a \leq Dx \leq b$. By mathematical strict monotonicity of square roots on $\mathbb{R}_{\geq 0}$, $\sqrt{a/D} \leq \sqrt{x} \leq \sqrt{b/D}$. Since $p^2 \leq aD$, dividing both sides by $D^2 > 0$ yields $(p/D)^2 \leq a/D$. Since $p \geq 0$, extracting the principal square root yields $p/D \leq \sqrt{a/D} \leq \sqrt{x}$. Since $bD \leq q^2$, dividing by D^2 yields $b/D \leq (q/D)^2$. Since $q \geq 0$, $\sqrt{x} \leq \sqrt{b/D} \leq q/D$. Thus $\sqrt{x} \in \gamma_D(p, q)$. □

7.5. Non-Linear Primitives (ReLU)

$$f^{\text{impl}}_{\text{relu}}(x) = \max(0, x).$$

Lemma 23 (ReLU Branch Covering and Mutual Exclusion). *For any $a, b \in \mathbb{Z}$ subject to $a \leq b$, exactly one of the following ground conditions mathematically holds: (1) $b \leq 0$, (2) $0 \leq a$, (3) $a < 0 \wedge 0 < b$.*

Proof. Applying the Trichotomy of integers to b against 0: If $b \leq 0$, case (1) holds. Since $a \leq b$, $a \leq 0$, excluding (2) and (3). If $0 < b$, we apply Trichotomy to a against 0: If $0 \leq a$, case (2) holds, excluding (3). If $a < 0$, then $a < 0 \wedge 0 < b$, satisfying case (3) and excluding (2). □

Witness: $W_{\text{relu}} = (A, B) \in \mathbb{Z}^2$.
Check Definition (Full Expansion): Instead of implicit control flow, the Verifier evaluates the absolute flat proposition:

$$\begin{aligned}\text{Check}_{\text{relu}}(a, b, A, B) := &(b \leq 0 \wedge A = 0 \wedge B = 0) \\ &\vee (0 \leq a \wedge A = a \wedge B = b) \\ &\vee (a < 0 \wedge 0 < b \wedge A = 0 \wedge B = b)\end{aligned}$$

Closed Form: $T_{\text{relu}}(a, b, A, B) := (A, B)$.

Theorem 6 (R-SOUND for relu). *T_{relu} satisfies R-SOUND.*

Proof. By Lemma 17, Abstract Domain Safety trivially holds. By Lemma 23, $\mathrm{WF}^{*}_{\mathrm{relu}}$ guarantees exactly one disjoint branch passes evaluation. Closure $A \leq B$ is trivial in all three. Case 1 ($b \leq 0$): For $x \in [a/D, b/D]$, $x \leq 0$, so $\max(0, x) = 0 \in [0, 0] = \gamma_D(A, B)$. Case 2 ($0 \leq a$): For $x \in [a/D, b/D]$, $x \geq 0$, so $\max(0, x) = x \in [a/D, b/D] = \gamma_D(A, B)$. Case 3 ($a < 0 \wedge 0 < b$): For $x \in [a/D, b/D]$, $x \leq b/D$. Also $\max(0, x) \geq 0$. If $x < 0$, $\max(0, x) = 0 \leq b/D$. If $x \geq 0$, $\max(0, x) = x \leq b/D$. In both mathematical sub-domains, $\max(0, x) \in [0, b/D] = \gamma_D(A, B)$. □

8. Constants, Program Model, and Continuous Trajectories

To completely eliminate mathematical hazard, theoretical specifications must compile precisely into integer constraints without implicit approximation loss.

Definition 9 (Spec Constant Set and Encoded Constants). *Let $K_{spec} \subset \mathbb{Z}$ be the finite set of all integer constant payloads appearing in the certificate's boundary inputs, program constants, and specification constraints. Each $k \in K_{spec}$ represents the exact rational constant $k/D \in \mathbb{Q} \subset \mathbb{R}$. No additional divisibility condition is required by the verifier; correctness is with respect to these encoded rationals.*

Theorem 7 (Exact Constant Embedding). *For any $k \in K_{spec}$, $\gamma_D(k, k) = \{k/D\}$. In particular, encoded constants embed into the integer domain with zero approximation error.*

Proof. Immediate from the definition $\gamma_D(a, b) = [a/D, b/D]$. □

Definition 10 (Program DAG). *A program $\mathcal{P}$ is a graph $V = \{1, \ldots, N\}$ with a typing function $t : V \to \{IN, CONST, OP\}$. IN nodes specify absolute boundary bounds $U_i \in \mathcal{I}_D$. CONST nodes declare an encoded integer constant $k_i \in K_{spec}$, mapped rigidly to $C_i = (k_i, k_i) \in \mathcal{I}_D$ and interpreted in real semantics as k_i/D. OP nodes specify $\sigma_i \in \Sigma_{prim}$ and parent indices $\pi(i) = (j_1, \ldots, j_k)$ enforcing topological strict order $j_m < i$.*

Definition 11 (Admissible Inputs and Trajectories). *Let $I_{in} = \{i \in V \mid t(i) = IN\}$. An **admissible input** is a mapping $u : I_{in} \to \mathbb{R}$ restricted strictly to $u(i) \in \gamma_D(U_i)$. A **concrete trajectory** is an execution sequence mapping $x : V \to \mathbb{R}$ generated topologically from u via the following partial functions:*

1. $t(i) = IN \implies x(i) = u(i)$
2. $t(i) = CONST \implies x(i) = k_i/D$
3. $t(i) = OP \implies$ *evaluated exclusively if* $(x(j_1), \ldots, x(j_k)) \in \mathrm{Dom}^{\mathbb{R}}_{\sigma_i}$, *yielding* $x(i) = f^{impl}_{\sigma_i}(x(j_1), \ldots, x(j_k))$.

9. Specification Compilation and Spec-REPLAY

To mathematically decouple the Verifier logic from recursive real-valued evaluation mechanics, specifications are algorithmically translated into Spec-DAG ledgers structurally isomorphic to program topology.

Definition 12 (Specification Syntactic Structure and Real Semantics $[\![e]\!]_{\mathbb{R}}$). *Each e is uniquely generated by $e ::= \mathsf{Const}(k) \mid \mathsf{Var}(v) \mid e_1 \oplus e_2$ where $k \in K_{spec}$, $v \in V$, and $\oplus \in \{+, -, \times\}$. For a totally defined trajectory x, the real continuous evaluation is defined structurally: $[\![\mathsf{Const}(k)]\!]_{\mathbb{R}}(x) := k/D$, $[\![\mathsf{Var}(v)]\!]_{\mathbb{R}}(x) := x(v)$, $[\![e_1 \oplus e_2]\!]_{\mathbb{R}}(x) := [\![e_1]\!]_{\mathbb{R}}(x) \oplus [\![e_2]\!]_{\mathbb{R}}(x)$.*

Definition 13 (Spec-DAG Compilation Mapping). *We define a completely formalised map* $\text{compile}(e)$ *yielding a Spec-DAG evaluated over* Σ_{prim}*:*

1. $\text{compile}(\mathsf{Const}(k))$ *returns a CONST node with payload* k*.*
2. $\text{compile}(\mathsf{Var}(v))$ *returns a VAR node referencing program node* v*.*
3. $\text{compile}(e_1 \oplus e_2)$ *returns an* $OP(\oplus)$ *node with parents* $\text{compile}(e_1)$ *and* $\text{compile}(e_2)$*.*

No sub-expression sharing is performed: compile *produces a tree-shaped DAG.*

Definition 14 (Spec-DAG Real Semantics val_{spec}). *Let* $G_e = \text{compile}(e)$ *and fix a valid trajectory* x*. Define* $\text{val}_{spec}(k, x)$ *by topological recursion on nodes* k *of* G_e*:*

1. *If* k *is CONST with payload* p*, then* $\text{val}_{spec}(k, x) := p/D$*.*
2. *If* k *is VAR referencing* $v \in V$*, then* $\text{val}_{spec}(k, x) := x(v)$*.*
3. *If* k *is* $OP(\oplus)$ *with parents* (k_1, k_2)*, then* $\text{val}_{spec}(k, x) := \text{val}_{spec}(k_1, x) \oplus \text{val}_{spec}(k_2, x)$*.*

Theorem 8 (Compile Correctness). *For any continuous specification expression* e *and valid trajectory* x*,* $\text{val}_{spec}(\text{compile}(e), x) = [\![e]\!]_{\mathbb{R}}(x)$*.*

Proof. We proceed by structural induction on the grammar of expressions e. *Base (Const).* If $e = \mathsf{Const}(k)$, then $\text{compile}(e)$ is a CONST node with payload k. By Definition of val_{spec}, $\text{val}_{\text{spec}}(\text{compile}(e), x) = k/D$. By Definition of $[\![\cdot]\!]_{\mathbb{R}}$, $[\![\mathsf{Const}(k)]\!]_{\mathbb{R}}(x) = k/D$. Hence equal. *Base (Var).* If $e = \mathsf{Var}(v)$, then $\text{compile}(e)$ is a VAR node referencing v. Thus $\text{val}_{\text{spec}}(\text{compile}(e), x) = x(v) = [\![\mathsf{Var}(v)]\!]_{\mathbb{R}}(x)$. *Step.* If $e = e_1 \oplus e_2$, then $\text{compile}(e)$ is an $\text{OP}(\oplus)$ node with parents $\text{compile}(e_1)$ and $\text{compile}(e_2)$. By definition,

$$\text{val}_{\text{spec}}(\text{compile}(e), x) = \text{val}_{\text{spec}}(\text{compile}(e_1), x) \oplus \text{val}_{\text{spec}}(\text{compile}(e_2), x).$$

By induction hypothesis this equals $[\![e_1]\!]_{\mathbb{R}}(x) \oplus [\![e_2]\!]_{\mathbb{R}}(x) = [\![e]\!]_{\mathbb{R}}(x)$. □

Definition 15 (Spec-ledger REPLAY). *The Spec-DAG is serialised topologically into a sequential Spec-ledger. Crucially, every* $OP(\times)$ *node is supplied with an explicit Euclidean integer witness* $W_\times$*. The Verifier computes abstract bounds* $\tilde{I}_k^{spec}$*:*

- *VAR nodes directly import the abstract bounds* $\tilde{I}_v$ *outputted by the main Program REPLAY sequence.*
- *CONST nodes insert* $(\text{payload}, \text{payload})$*.*
- *OP nodes rigorously invoke* $\tau(\text{Check}_\sigma^*)$ *followed sequentially by bounding* T_σ*.*

10. Simultaneous Soundness and Absolute Verification

Definition 16 (Specification Constraint Index Set Φ and Targets). *Let* $\Phi \subset \mathbb{N}$ *be a finite index set. A specification constraint family is a family* $(e_r)_{r\in\Phi}$ *of expressions generated by the grammar* $e ::= \mathsf{Const}(k) \mid \mathsf{Var}(v) \mid e_1 \oplus e_2$ *with* $k \in K_{spec}$*,* $v \in V$*, and* $\oplus \in \{+, -, \times\}$*. Each* $r \in \Phi$ *denotes the real inequality constraint* $[\![e_r]\!]_{\mathbb{R}}(x) \le 0$ *imposed on a trajectory* x*. Define the Spec-root node* $k_r := \text{compile}(e_r)$*.*

Definition 17 (Well-Formed Certificate). *A certificate* $\mathcal{C} = (\mathcal{P}, U, C, \Phi, \mathcal{L}_{prog}, \mathcal{L}_{spec})$ *is* well-formed *if all of the following hold:*

1. ***(Scale)*** $0 < D$*.*
2. ***(Interval Closure)*** *Every interval appearing in* U*,* C*, and in the program ledger assignments* $(I_i)_{i\in V}$ *satisfies* $a \le b$*.*
3. ***(Program Structure)*** $\mathcal{P}$ *is a DAG with parent indices satisfying* $j \in \pi(i) \implies j < i$*.*

https://doi.org/10.3390/math1010000

4. (***Specification Targets***) Φ *is finite and each* e_r $(r \in \Phi)$ *is generated by the specification grammar over* (K_{spec}, V)*; moreover,* $\mathcal{L}_{spec}$ *is a topological serialization of the disjoint union of the compiled trees* $(\mathrm{compile}(e_r))_{r\in\Phi}$.

Definition 18 (Certificate and Universal Verifier Algorithm). *A certificate*

$$\mathcal{C} = (\mathcal{P},\ U,\ C,\ \Phi,\ \mathcal{L}_{\mathrm{prog}},\ \mathcal{L}_{\mathrm{spec}})$$

maps topology onto I_i *intervals and* W_i *witnesses.*

The algorithm

$$\mathcal{V}(\mathcal{C}) \in \{\mathrm{ACCEPT}, \mathrm{REJECT}\}$$

executes exclusively over instantiated ground quantifier-free formulas over Σ_{int}.

Certificate-side check schemata may use Σ_{src} *but are normalized by* τ *before evaluation.*

1. (***WF-CERT***) *Verify Definition 17 for all certificate-provided objects, including finiteness of* Φ *and well-formedness of each* e_r $(r \in \Phi)$.
2. (***STRUCT***) *For all* OP*, verify DAG topological order* $j \in \pi(i) \implies j < i$.
3. (***REPLAY***) *Trace topologically to define* $\tilde{I}_i$. *IN sets* U_i. *CONST sets* C_i. *OP evaluates* $\tau(\mathrm{Check}^*_{\sigma_i}(\tilde{I}_{\pi(i)}, W_i))$*; if true, executes* $T_{\sigma_i}(\dots)$ *to generate* $\tilde{I}_i$. *Ensure* $I_i = \tilde{I}_i$.
4. (***SPEC-REPLAY***) *Execute Definition 15 across* $\mathcal{L}_{spec}$ *to compute* $\tilde{I}^{\mathrm{spec}}_k$ *for all Spec-ledger nodes* k*; for each* $r \in \Phi$*, let* $k_r := \mathrm{compile}(e_r)$ *and set* $(a_r, b_r) := \tilde{I}^{\mathrm{spec}}_{k_r}$.
5. (***SPEC-VERIFY***) *For each* $r \in \Phi$*, evaluate* $\tau(b_r \le 0)$.

Lemma 24 (REPLAY establishes WF*). *Because the Verifier REPLAY logic explicitly requires the strict truth of the unquantified evaluation* $\tau(\mathrm{Check}^*_{\sigma_i})$*, successful processing mechanically instantiates* $\mathrm{WF}^*_{\sigma_i}$ *by definition.*

Lemma 25 (WF* implies Domain Indicator). *For any* σ*, if* $\tau(\mathrm{Check}^*_\sigma(\cdot))$ *is true then* $\tau(\mathrm{Dom}_\sigma(\cdot))$ *is true.*

Proof. Immediate from $\mathrm{Check}^*_\sigma := \mathrm{Dom}_\sigma \wedge \mathrm{Check}_\sigma$. □

Theorem 9 (Simultaneous Trajectory Existence and Program Enclosure). *If* $\mathcal{V}(\mathcal{C}) =$ ACCEPT*, then for every admissible continuous input* u*, there exists a mathematically defined continuous trajectory* $x : V \to \mathbb{R}$ *such that across all topological program nodes* i*,* $x(i) \in \gamma_D(\tilde{I}_i)$.

Proof. We proceed by induction on the topological order of the program DAG.

Base case (IN, CONST): If $t(i) = \mathrm{IN}$, define $x(i) := u(i)$. By admissibility of the input, $u(i) \in \gamma_D(U_i) = \gamma_D(\tilde{I}_i)$. If $t(i) = \mathrm{CONST}$, define $x(i) := k_i/D$. Then $x(i) \in \gamma_D(C_i) = \gamma_D(\tilde{I}_i)$.

Inductive step (OP node i): Assume that for every parent $j \in \pi(i)$, the value $x(j)$ is already defined and satisfies $x(j) \in \gamma_D(\tilde{I}_j)$. Since REPLAY accepts node i, Lemma 24 yields $\mathrm{WF}^*_{\sigma_i}$. By Lemma 25, $\tau(\mathrm{Dom}_{\sigma_i}(\tilde{I}_{\pi(i)}))$ is true. Applying the corresponding domain-safety lemma for σ_i gives

$$\gamma_D(\tilde{I}_{j_1}) \times \cdots \times \gamma_D(\tilde{I}_{j_k}) \subseteq \mathrm{Dom}^{\mathbb{R}}_{\sigma_i}.$$

Since each parent value lies in the corresponding enclosure, the tuple $x(\pi(i))$ belongs to $\mathrm{Dom}^{\mathbb{R}}_{\sigma_i}$. Therefore $f^{\mathrm{impl}}_{\sigma_i}(x(\pi(i)))$ is defined; set

$$x(i) := f^{\mathrm{impl}}_{\sigma_i}(x(\pi(i))).$$

Because $\mathrm{WF}^*_{\sigma_i}$ holds, the corresponding R-SOUND theorem applies and yields

$$x(i) \in (f^{\mathrm{impl}}_{\sigma_i})^\sharp(\gamma_D(\tilde{I}_{\pi(i)})) \subseteq \gamma_D(T_{\sigma_i}(\tilde{I}_{\pi(i)}, W_i)).$$

By the REPLAY definition, $T_{\sigma_i}(\tilde{I}_{\pi(i)}, W_i) = \tilde{I}_i$. Hence $x(i) \in \gamma_D(\tilde{I}_i)$.

Thus existence and enclosure hold at every node. The claim follows by induction. □

Lemma 26 (Uniqueness of Trajectory). *Assume $\mathcal{V}(\mathcal{C}) = \mathrm{ACCEPT}$. For a fixed admissible input u, if $x : V \to \mathbb{R}$ and $x' : V \to \mathbb{R}$ are trajectories satisfying Definition 11 for the same u, then $x = x'$ pointwise.*

Proof. By topological induction on node index i. If $t(i) = \mathrm{IN}$, then $x(i) = u(i) = x'(i)$. If $t(i) = \mathrm{CONST}$, then $x(i) = c_i = x'(i)$. If $t(i) = \mathrm{OP}$, by induction hypothesis $x(j) = x'(j)$ for all $j \in \pi(i)$. Hence the parent tuples are identical, and both trajectories must satisfy

$$x(i) = f^{\mathrm{impl}}_{\sigma_i}(x(\pi(i))) \quad \text{and} \quad x'(i) = f^{\mathrm{impl}}_{\sigma_i}(x'(\pi(i))).$$

Therefore $x(i) = x'(i)$. This completes induction. □

Theorem 10 (Spec-DAG Enclosure). *Assume $\mathcal{V}(\mathcal{C}) = \mathrm{ACCEPT}$. For any admissible input u and the induced trajectory x from Theorem 9, every Spec-ledger node k satisfies*

$$\mathrm{val}_{spec}(k, x) \in \gamma_D(\tilde{I}^{spec}_k).$$

Proof. Proceed by topological induction over the Spec-ledger order used by SPEC-REPLAY (Definition 15). Let k be the current node and assume the claim holds for all strict parents of k in the Spec-DAG.

Case 1 (CONST). Then $\tilde{I}^{\mathrm{spec}}_k = (p, p)$ where p is the payload integer. By Definition of $\mathrm{val}_{\mathrm{spec}}$, $\mathrm{val}_{\mathrm{spec}}(k, x) = p/D \in [p/D, p/D] = \gamma_D(p, p) = \gamma_D(\tilde{I}^{\mathrm{spec}}_k)$.

Case 2 (VAR). Then k references some program node v, and SPEC-REPLAY sets $\tilde{I}^{\mathrm{spec}}_k := \tilde{I}_v$. Also $\mathrm{val}_{\mathrm{spec}}(k, x) = x(v)$ by definition. By Theorem 9, $x(v) \in \gamma_D(\tilde{I}_v) = \gamma_D(\tilde{I}^{\mathrm{spec}}_k)$.

Case 3 (OP). Let k be $\mathrm{OP}(\sigma)$ with Spec-parents $k_1, \ldots, k_m$. By induction hypothesis,

$$\mathrm{val}_{\mathrm{spec}}(k_j, x) \in \gamma_D(\tilde{I}^{\mathrm{spec}}_{k_j}) \qquad (j = 1, \ldots, m).$$

By Definition 15, SPEC-REPLAY evaluates the ground formula

$$\tau\Big(\mathrm{Check}^*_\sigma(\tilde{I}^{\mathrm{spec}}_{k_1}, \ldots, \tilde{I}^{\mathrm{spec}}_{k_m}, W_k)\Big).$$

and ACCEPT implies it evaluates to true. Therefore WF^*_σ holds for this Spec-node. By Abstract Domain Safety for σ (Lemma 17 for $\sigma \in \{+, -, \times, \mathrm{relu}\}$, Lemma 15 for $\sigma = \mathrm{inv}$, and Lemma 16 for $\sigma = \mathrm{sqrt}$),

$$\gamma_D(\tilde{I}^{\mathrm{spec}}_{k_1}) \times \cdots \times \gamma_D(\tilde{I}^{\mathrm{spec}}_{k_m}) \subseteq \mathrm{Dom}^{\mathbb{R}}_\sigma.$$

Hence the real operation f^{impl}_σ is defined on every input point in that product set, in particular on the tuple $(\mathrm{val}_{\mathrm{spec}}(k_1, x), \ldots, \mathrm{val}_{\mathrm{spec}}(k_m, x))$. By R-SOUND (Definition 8), SPEC-REPLAY sets $\tilde{I}^{\mathrm{spec}}_k = T_\sigma(\tilde{I}^{\mathrm{spec}}_{k_1}, \ldots, \tilde{I}^{\mathrm{spec}}_{k_m}, W_k)$ and we have

$$(f^{\mathrm{impl}}_\sigma)^\sharp(\gamma_D(\tilde{I}^{\mathrm{spec}}_{k_1}), \ldots, \gamma_D(\tilde{I}^{\mathrm{spec}}_{k_m})) \subseteq \gamma_D(\tilde{I}^{\mathrm{spec}}_k).$$

Applying this inclusion to the particular tuple yields

$$\mathrm{val}_{\mathrm{spec}}(k, x) = f_\sigma^{\mathrm{impl}}\big(\mathrm{val}_{\mathrm{spec}}(k_1, x), \dots\big) \in \gamma_D(\tilde{I}_k^{\mathrm{spec}}).$$

All cases are closed, completing the induction. □

Theorem 11 (Deterministic Verifier Soundness)**.** *If $\mathcal{V}(\mathcal{C}) = \mathrm{ACCEPT}$, then for any admissible input u, letting x be the induced trajectory guaranteed by Theorem 9, we have*

$$\forall r \in \Phi,\ [\![e_r]\!]_{\mathbb{R}}(x) \le 0.$$

Proof. Fix $r \in \Phi$ and let $k_r := \mathrm{compile}(e_r)$. By Spec-DAG Enclosure,

$$\mathrm{val}_{\mathrm{spec}}(k_r, x) \in \gamma_D(\tilde{I}_{k_r}^{\mathrm{spec}}).$$

By SPEC-REPLAY, $\tilde{I}_{k_r}^{\mathrm{spec}} = (a_r, b_r)$, hence

$$\mathrm{val}_{\mathrm{spec}}(k_r, x) \in \gamma_D(a_r, b_r) = [a_r/D, b_r/D].$$

By Compile Correctness (Theorem 8),

$$[\![e_r]\!]_{\mathbb{R}}(x) = \mathrm{val}_{\mathrm{spec}}(k_r, x) \in [a_r/D, b_r/D].$$

The ACCEPT sequence demanded $\tau(b_r \le 0)$, hence $b_r \le 0$ by Lemma 9. Consequently,

$$[\![e_r]\!]_{\mathbb{R}}(x) \le b_r/D \le 0.$$

Since $r \in \Phi$ was arbitrary, the claim holds for all $r \in \Phi$. □

10.1. Deterministic Complexity and Implementation Bound Integration

Definition 19 (Atomic Evaluation Cost)**.** *We measure the cost of evaluating a normalized check $\tau(\phi)$ by the total number of integer comparisons ($<, \le, =$) evaluated in the worst case, assuming no short-circuiting for $\wedge$ and $\vee$.*

Constraint 1 (Atomic Boundedness of Primitives)**.** *Let an atomic operation uniquely map to an exact* `Atom` *comparison ($\le, <, =$). By formally translating macro rules into totally expanded Σ_{int} evaluations, the count of evaluated atoms k_σ structurally stabilizes:*

- $k_+, k_- = 0$.
- $\mathrm{Check}_{FDIV}, \mathrm{Check}_{CDIV} = 6$ *atoms each.*
- $\mathrm{Check}_{\min}, \mathrm{Check}_{\max} = 8$ *atoms each.*
- $k_\times = 8 + 8 + 6 + 6 = 28$ *atoms.*
- $k_{inv} = 2 + 6 + 6 = 14$ *atoms.*
- $k_{sqrt} = 1 + 1 + 1 + 1 + 1 + 2 = 7$ *atoms.*
- $k_{relu} = 3 + 3 + 4 = 10$ *atoms.*

Thus $k = \max(k_\sigma) \le 28$ maps logic geometrically to an absolute uniform processing threshold.

Theorem 12 (Deterministic Linear-Time Complexity)**.** *Let $n = |\mathcal{L}_{prog}|$ and $s = |\mathcal{L}_{spec}|$. Define metric b unconditionally as the absolute maximal bit-length instantiated during all discrete verifier integer generations, including completely internal algebraic multiplications (e.g., intermediate terms $ac, (Q+1)d, p^2$). The REPLAY algorithm triggers structurally at maximum $k \le 28$ atomic bounds*

per trace node. Marking $C(b)$ as the multi-precision timing index, global execution completes totally bounded within:

$$T_{verify}(n, s, b) = O((n + s) \cdot C(b))$$

Assumption 1 (Implementation Inclusion / Deployment Contract)**.** *For each primitive $\sigma \in \Sigma_{prim}$, let the deployed implementation be denoted by g_σ. Assume the following deployment contract holds for every invocation of σ in any trace $\mathcal{C}$ for which $\mathcal{V}(\mathcal{C}) =$ ACCEPT:*

1. *(**Totality on verifier-enclosed tuples**) Whenever the Verifier accepts, every runtime argument tuple $\vec{x}$ used to call g_σ lies in the verifier-produced enclosure product*

$$\vec{x} \in \gamma_D(I_1) \times \cdots \times \gamma_D(I_k) \subseteq \mathrm{Dom}_\sigma^{\mathbb{R}},$$

   and $g_\sigma(\vec{x})$ returns a real value for all such $\vec{x}$ (i.e., no NaN/Inf or undefined branch on this set).
2. *(**Inclusion into the certified set-extension**) For every such tuple $\vec{x}$,*

$$g_\sigma(\vec{x}) \in (f_\sigma^{impl})^\sharp(\gamma_D(I_1), \ldots, \gamma_D(I_k)).$$

Equivalently: on each verifier-enclosed region, the deployed g_σ refines the abstract set-extension used in the soundness rules, so the enclosure guarantees transfer from f_σ^{impl} to the deployed g_σ under the stated contract.

Corollary 1 (Transfer to Deployed Implementations under Implementation Inclusion)**.** *The core theorems certify enclosure for the mathematical semantics f_σ^{impl} and the verifier-produced integer envelopes. Under Assumption 1, the same enclosure guarantee transfers to the deployed implementation g_σ on the certified region.*

11. Discussion and Limitations

Interpretation of the present results.

The present results should be read as closure theorems for the verifier core over the fixed primitive set Σ_{prim}. Relative to classical interval analysis, the contribution is not a new interval semantics. Relative to proof-carrying code, it is a specialization of the proof object to finite instantiated integer checks with explicit Euclidean witnesses. Relative to floating-point tools and SMT-based reasoning, it is the elimination of search and analytic reasoning from the verifier itself. Accordingly, the paper proves deterministic replay-checkability of realized certificates for the stated primitive core; it does not claim a general result for arbitrary nonlinear-real verification or for arbitrary deployed implementations outside the explicit implementation-inclusion contract.

On Spec-DAG sharing and what "blow-up" means here.

Our specification compilation intentionally performs no sub-expression sharing, producing a tree-shaped Spec-DAG by definition. This is a certificate-generation design choice (and can be optimized by sharing), but it is not a verifier requirement. Accordingly, we do not claim to prevent syntactic expansion of the certificate in general; rather, we avoid solver-driven blow-up from non-linear real arithmetic encodings by restricting verification to ground Σ_{int} checks and replaying a finite ledger. The deterministic bound $O((n + s)\, C(b))$ is stated in terms of the realized ledger sizes $n = |\mathcal{L}_{\mathrm{prog}}|$ and $s = |\mathcal{L}_{\mathrm{spec}}|$.

Deployment binding (Implementation Inclusion).

The core theorems certify enclosure for the mathematical semantics f_σ^{impl} and the verifier-produced integer envelopes. Transfer of this guarantee to a deployed implementation g_σ is *not* claimed unconditionally. It holds only under Assumption 1, which requires

(i) runtime arguments to remain within the verifier-enclosed product region, and (ii) the deployed g_σ to refine the certified set-extension on that region. If this contract is violated (e.g., implementation drift, undefined branches, NaN/Inf behavior, or domain mismatch), the enclosure guarantee does not apply to the deployed device.

Scope of primitives and extensibility.

This work closes soundness for the finite primitive set

$$\Sigma_{\text{prim}} = \{+, -, \times, \text{inv}, \text{sqrt}, \text{relu}\}.$$

Any additional primitive (or any alternative implementation contract) requires an explicit check schema and a corresponding R-SOUND proof in the same ground, quantifier-free integer framework.
Therefore, the claimed determinism and closure are scoped to the explicitly discharged primitives and their stated domains.

Numeric model and exactness requirements.

Verification is performed over mathematical integers with bit-complexity accounted for by $C(b)$. A concrete verifier implementation must therefore ensure exact integer semantics (e.g., via big-integer arithmetic) and must not rely on finite-word overflow behavior. Any mismatch between mathematical integer evaluation and the implementation arithmetic invalidates the replay-verification guarantee.

Certificate generation versus replay verification.

The deterministic bound $O((n+s)\cdot C(b))$ applies to *replay verification* of a fixed ledger and its witnesses. Certificate generation (ledger construction and witness synthesis) is intentionally treated as upstream work and may be substantially more expensive depending on the compilation and bounding strategy. The contribution of ADIC is that, once produced, verification is finite, deterministic, and mechanically checkable.

Provenance and adversarial artefacts.

The verifier rejects any ledger/witness set whose ground integer checks fail; however, provenance protection (e.g., distribution integrity, signing, or binding to a build identity) is orthogonal to the mathematical core and is not required for the correctness of the replay-verification definition itself. If provenance guarantees are needed in deployment, they can be layered independently without changing the proof obligations of the verifier.

12. Conclusions

This paper establishes a verifier-closure result for replay verification of interval programs over the fixed primitive set $\Sigma_{\text{prim}} = \{+, -, \times, \text{inv}, \text{sqrt}, \text{relu}\}$. The contribution is not a new family of interval operators and not a general result for nonlinear-real verification. Within the stated primitive set and assumptions, the verifier reduces checking to instantiated ground integer formulas over finite realized ledgers, and verifier acceptance implies the existence of a concrete trajectory together with enclosure of the certified specification constraints. The deterministic complexity claim applies to replay verification of a realized certificate, up to integer bit-complexity. Extension to additional primitives requires the corresponding check schemas and R-SOUND proofs, and transfer to deployed floating-point implementations requires the explicit implementation-inclusion contract.

Supplementary Materials: Not applicable.

https://doi.org/10.3390/math1010000

Author Contributions: Conceptualization, H.M.; methodology, H.M.; formal analysis, H.M.; investigation, H.M.; writing–original draft preparation, H.M.; writing–review and editing, H.M.

Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: A public companion Lean proof artifact formalizing a core lemma of the ADIC verifier architecture is available at https://ghostdrifttheory.github.io/adic-lean-proof/. A public companion ADIC-based electricity-demand audit demonstration with published certificates and an audit report is available at https://github.com/GhostDriftTheory/ghostdrift-adic-audit/tree/main. These materials are supplementary to the present theoretical paper and are not required for verification of the formal results proved here.

Acknowledgments: During the preparation of this manuscript, the author used ChatGPT by OpenAI (accessed 2026-03-06) for limited language polishing and LaTeX formatting assistance. The author reviewed and edited all generated text and takes full responsibility for the content of this publication.

Conflicts of Interest: The author declares no conflict of interest.

Abbreviations

ADIC	Analytically Derived Interval Computation Integrity Certificates
TCB	Trusted Computing Base
DAG	Directed Acyclic Graph
Spec-DAG	Specification Directed Acyclic Graph
QF	Quantifier-Free
SMT	Satisfiability Modulo Theories

References

1. IEEE. IEEE Standard for Floating-Point Arithmetic. *IEEE Std 754-2019* **2019**, 1–84.
2. Daumas, M.; Melquiond, G. Certification of bounds on expressions involving rounded operators. *ACM Transactions on Mathematical Software (TOMS)* **2010**, *37*, 1–20.
3. Solovyev, A.; Borges, C.; Rangel, M. S.; Alnaasan, M. S.; Rakamarić, Z. Rigorous Estimation of Floating-Point Round-Off Errors with Symbolic Taylor Expansions. *ACM Transactions on Programming Languages and Systems (TOPLAS)* **2018**, *41*, 1–39.
4. Titolo, L.; Moscato, M. M.; Feliu, M. A.; Masci, P.; Muñoz, C. A. Rigorous Floating-Point Round-Off Error Analysis in PRECiSA 4.0. In *Formal Methods*; Springer: Cham, Switzerland, 2024; pp. 20–38. doi:10.1007/978-3-031-71177-0_2
5. Appel, A. W.; Kellison, A. VCFloat2: Floating-Point Error Analysis in Coq. In *Proceedings of the 13th ACM SIGPLAN International Conference on Certified Programs and Proofs (CPP 2024)*; ACM: New York, NY, USA, 2024; pp. 14–29. doi:10.1145/3636501.3636953.
6. Faissole, F.; Geneau de Lamarlière, P.; Melquiond, G. End-To-End Formal Verification of a Fast and Accurate Floating-Point Approximation. In *15th International Conference on Interactive Theorem Proving (ITP 2024)*; Leibniz International Proceedings in Informatics (LIPIcs), Vol. 309; Schloss Dagstuhl–Leibniz-Zentrum für Informatik: Dagstuhl, Germany, 2024; pp. 14:1–14:18. doi:10.4230/LIPIcs.ITP.2024.14.
7. Necula, G. C. Proof-carrying code. In *Proceedings of the 24th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL '97)*; ACM: New York, NY, USA, 1997; pp. 106–119.
8. IEEE. IEEE Standard for Interval Arithmetic. *IEEE Std 1788-2015* **2015**, 1–97.
9. Moore, R. E. *Interval Analysis*; Prentice-Hall: Englewood Cliffs, NJ, USA, 1966.
10. Neumaier, A. Interval Methods for Systems of Equations. *Acta Numerica* **2020**, *29*, 1–134.
11. Brucker, A. D.; Cameron-Burke, T.; Stell, A. Formally Verified Interval Arithmetic and Its Application to Program Verification. In *Proceedings of the 2024 IEEE/ACM 12th International Conference on Formal Methods in Software Engineering (FormaliSE '24)*; ACM: New York, NY, USA, 2024; pp. 111–121. doi:10.1145/3644033.3644370.

12. Cousot, P.; Cousot, R. Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Proceedings of the 4th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL '77)*; ACM: New York, NY, USA, 1977; pp. 238–252.
13. Desmartin, R.; Isac, O.; Passmore, G.; Komendantskaya, E.; Stark, K.; Katz, G. A Certified Proof Checker for Deep Neural Network Verification in Imandra. In *16th International Conference on Interactive Theorem Proving (ITP 2025)*; Leibniz International Proceedings in Informatics (LIPIcs), Vol. 352; Schloss Dagstuhl–Leibniz-Zentrum für Informatik: Dagstuhl, Germany, 2025; pp. 1:1–1:21. doi:10.4230/LIPIcs.ITP.2025.1.
14. de Moura, L.; Bjørner, N. Z3: An Efficient SMT Solver. In *Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*; Springer: Berlin/Heidelberg, Germany, 2008; pp. 337–340.
15. Kremer, G.; Reynolds, A.; Barrett, C.; Tinelli, C. Cooperating Techniques for Solving Nonlinear Real Arithmetic in the cvc5 SMT Solver (System Description). In *Proceedings of the 11th International Joint Conference on Automated Reasoning (IJCAR '22)*; Lecture Notes in Computer Science, Vol. 13385; Springer Nature: Cham, Switzerland, 2022; pp. 95–105. doi:10.1007/978-3-031-10769-6_7