Lazy Ultrafilters for Hyperreal Computation: Theory and Implementation of Non-Constructive Mathematics

Krzysztof Woś
The University of Tokyo
Graduate School of Information Science and Technology
Department of Creative Informatics

Abstract

Hyperreal numbers, which extend the reals with infinitesimals and infinite quantities, provide elegant foundations for analysis but are built from non-constructive objects (non-principal ultrafilters) whose existence depends on the Axiom of Choice. We present a practical implementation of hyperreal arithmetic through a lazy partial ultrafilter construction that commits only to constraints demanded by program execution. Comparisons are resolved by deterministic finite-or-cofinite and series-based reasoning when possible, and otherwise by incremental SAT solving. We also implement standard part extraction for a fragment of analytic functions via Laurent-series recognition in $\delta = 1/n$. In a benchmark suite of 40 membership queries, 19 (47.5%) are resolved without SAT via finite-or-cofinite reasoning and memoization, concentrating SAT effort on genuinely choice-dependent oscillatory sequences.

Keywords. Nonstandard analysis; ultrafilters; hyperreals; SAT; partial ultrafilter; standard part; programming language semantics; lazy evaluation.

1 Introduction

The hyperreal numbers, which rigorously formalize the intuitive notion of infinitesimals that Leibniz and Newton employed in developing calculus, have remained a mathematical curiosity rather than a computational tool. While Robinson's nonstandard analysis [16] established their rigorous foundations through ultrafilters and the transfer principle, the inherently non-constructive nature of these foundations has obstructed direct implementation. This paper presents a computational realization of ultrafilter-based hyperreal arithmetic, resolving key practical obstacles while preserving classical soundness relative to an ultrafilter completion.

The challenge in implementing hyperreals stems from a foundational paradox. Their construction requires non-principal ultrafilters on the natural numbers, mathematical objects whose existence depends on the Axiom of Choice [9]. No explicit construction of such an ultrafilter is known, and different ultrafilters yield different hyperreal fields. For instance, the truth of the comparison $[\sin(n)] > 0$ depends on whether the index set $\{n \in \mathbb{N} : \sin(n) > 0\}$ is selected by the ultrafilter. This non-constructive foundation appears to preclude any computational realization.

Our key insight is that finite computations require only finite commitments. Rather than attempting to construct an ultrafilter (maximal filter), we maintain a *partial ultrafilter* that we extend lazily as the program executes. When the program compares two hyperreal expressions, we determine whether existing constraints force a particular outcome. If not, we make a consistent choice and record it as a new constraint. This approach transforms the non-constructive choice of ultrafilter into a series of computational decisions made during program execution.

The theoretical foundation for this approach is a completion argument. A terminating run makes only finitely many comparison queries and thus refers to only finitely many index sets. We maintain these commitments as a satisfiable constraint system; by the ultrafilter lemma (via Zorn's lemma) and the fact that non-principal ultrafilters extend the Fréchet filter of cofinite sets, any consistent finite family of commitments that avoids finite sets extends to a non-principal ultrafilter. Therefore each finite run is consistent with some classical hyperreal field, even though no ultrafilter is explicitly constructed. Different completions may disagree on underdetermined comparisons, but they are elementarily equivalent by Łoś's theorem [12].

Our implementation realizes this theoretical framework through careful engineering. We maintain a partial ultrafilter as a conjunction of constraints, employing SAT solving for consistency checking when extending it. However, we identify a substantial decidable fragment where comparisons can be resolved without SAT solving, including all comparisons involving standard reals, the infinitesimal $1/n$, and the infinite hyperreal n. For standard part extraction, we implement a series recognizer that handles compositions of common analytic functions by computing Laurent series expansions in powers of $1/n$.

While the technical development is expressed in terms of operational semantics, constraint solving, and implementation, the motivating question is foundational: what entitles a finite computation to speak about continuum objects such as infinitesimals? Section 7 articulates our answer in terms of finite observation and classical completion; it is the conceptual core that motivates the technical development.

Contributions This paper makes four primary contributions to the intersection of nonstandard analysis and programming languages:

First, we develop a complete operational semantics for hyperreal computation based on lazy partial ultrafilters. Our semantics maintains mathematical soundness while requiring only finite computational resources for finite programs. We prove that our construction satisfies the essential properties of ultrafilters restricted to the finite set of comparisons made during execution.

Second, we give a completion-based soundness argument explaining why finite computations can safely interact with non-constructive objects like ultrafilters: when the runtime commitments remain consistent, they admit at least one classical completion.

Third, we implement a practical system for hyperreal computation with demonstrated efficiency. Our evaluation shows that fast paths and memoization resolve 47.5% of membership queries without SAT, while the remaining SAT calls concentrate on oscillatory or adversarial inputs. The system successfully handles diverse computational tasks including derivative calculation, limit evaluation, and series expansion.

Fourth, we formalize the relationship between hyperreal differentiation and automatic differentiation. We prove that these approaches coincide at smooth points while identifying precise conditions where they diverge, particularly at discontinuities and control flow branches. This analysis clarifies the distinct capabilities of each approach.

2 Background: Hyperreals and Ultrafilters

2.1 Mathematical Foundations

The hyperreal numbers extend the real numbers with infinitesimal and infinite quantities while preserving the algebraic and order structure of the reals. Their construction relies on ultrafilters, which provide a consistent way to extract limiting behavior from sequences.

Definition 1 (Ultrafilter)**.** *A* non-principal ultrafilter *on* $\mathbb{N}$ *is a collection* $\mathcal{U} \subseteq \mathcal{P}(\mathbb{N})$ *satisfying:*

1. ***Non-triviality****:* $\mathbb{N} \in \mathcal{U}$ *and* $\emptyset \notin \mathcal{U}$
2. ***Upward closure****: If* $A \in \mathcal{U}$ *and* $A \subseteq B \subseteq \mathbb{N}$*, then* $B \in \mathcal{U}$
3. ***Finite intersection****: If* $A, B \in \mathcal{U}$*, then* $A \cap B \in \mathcal{U}$
4. ***Maximality****: For every* $A \subseteq \mathbb{N}$*, exactly one of* A *or* $\mathbb{N} \setminus A$ *is in* $\mathcal{U}$
5. ***Non-principality****: No finite set is in* $\mathcal{U}$

Given such an ultrafilter $\mathcal{U}$, the hyperreal field ${}^*\mathbb{R}$ is constructed as the quotient of real sequences modulo the equivalence relation induced by $\mathcal{U}$. Two sequences (a_n) and (b_n) are equivalent when they agree on a set in $\mathcal{U}$, that is, when $\{n \in \mathbb{N} : a_n = b_n\} \in \mathcal{U}$.

The transfer principle, formalized by Łoś's theorem [12], establishes that first-order properties of $\mathbb{R}$ lift to ${}^*\mathbb{R}$. If ϕ is a first-order sentence in the language of ordered fields, then ϕ holds in $\mathbb{R}$ if and only if ϕ holds in ${}^*\mathbb{R}$. This principle ensures that ${}^*\mathbb{R}$ extends $\mathbb{R}$ conservatively, preserving all first-order truths about real numbers.

In classical nonstandard analysis, the standard part map st is defined on finite hyperreals: if $x \in {}^*\mathbb{R}$ is finite, there exists a unique $r \in \mathbb{R}$ such that $x - r$ is infinitesimal, and we write $\mathrm{st}(x) = r$. (Infinite hyperreals have no standard part.)

In our implementation, extracting $\mathrm{st}(x)$ from an arbitrary representative sequence can be completion-dependent. Accordingly, we expose a *partial* standard-part extractor $\widehat{\mathrm{st}} : {}^*\mathbb{R} \to \mathbb{R} \cup \{\perp\}$ (Sec. 4). It returns r when it can certify, in a completion-invariant decidable fragment (via series recognition), that x is near-standard with $\mathrm{st}(x) = r$, and returns $\perp$ otherwise.

2.2 The Constructivity Challenge

The existence of non-principal ultrafilters cannot be proved without the Axiom of Choice, and no explicit construction is known. This non-constructive foundation creates fundamental challenges for computational implementation. Different ultrafilters yield different hyperreal fields that, while elementarily equivalent, disagree on specific values. For instance, the sign of the hyperreal represented by the sequence $(\sin(n))_{n\in\mathbb{N}}$ depends entirely on the chosen ultrafilter, as does the limiting behavior of any non-convergent sequence.

Furthermore, the maximality requirement forces every subset of $\mathbb{N}$ to be decided, either included in or excluded from the ultrafilter. This requirement appears to demand infinite information, making direct implementation impossible. Any finite approximation seems doomed to encounter undecidable cases where neither inclusion nor exclusion is forced by existing constraints.

Previous approaches to computational nonstandard analysis have often abandoned ultrafilters entirely, working with syntactic representations of infinitesimals, or restricted attention to convergent sequences where ultrafilter choice becomes irrelevant. Our approach keeps the ultrafilter-based foundation while exposing a finite, executable interface via a lazily-extended partial ultrafilter.

3 The Lazy Partial Ultrafilter Construction

3.1 Core Idea: Incremental Commitment

Our construction resolves the constructivity paradox through a simple but powerful observation: a program that terminates makes only finitely many comparisons between hyperreal values. Rather than constructing an ultrafilter (maximal filter) upfront, we maintain a partial ultrafilter that grows incrementally as the program executes. When the program compares two hyperreals, we check whether existing constraints determine the result. If they do not, we make a consistent choice and add it to our constraints.

Formally, we maintain a partial ultrafilter $\mathcal{U}_k$ after k comparison operations, where $\mathcal{U}_k$ is a finite collection of subsets of $\mathbb{N}$ that satisfies the ultrafilter properties restricted to its members. The key invariant is that $\mathcal{U}_k$ can be extended to an ultrafilter (maximal filter)—that is, our choices remain consistent with the existence of a completion.

3.2 Representing Hyperreals

We represent hyperreals as expressions in a simple language of sequences. The base sequences include constants c for $c \in \mathbb{R}$, the identity sequence n, the reciprocal sequence $1/n$, and the alternating sequence $(-1)^n$. Complex sequences are built through arithmetic operations and analytic functions.

Each hyperreal comparison generates a partition of $\mathbb{N}$ into three sets. For hyperreals a and b, we define:

$$L_{a,b} = \{n \in \mathbb{N} : a_n < b_n\} \tag{1}$$
$$E_{a,b} = \{n \in \mathbb{N} : a_n = b_n\} \tag{2}$$
$$G_{a,b} = \{n \in \mathbb{N} : a_n > b_n\} \tag{3}$$

These sets form a partition: $L_{a,b} \cup E_{a,b} \cup G_{a,b} = \mathbb{N}$ and the sets are pairwise disjoint. The ultrafilter must contain exactly one of these three sets, determining whether $a < b$, $a = b$, or $a > b$ in the hyperreal field.

3.3 Maintaining Consistency

The partial ultrafilter must satisfy several consistency requirements. First, complementary sets must have opposite membership: if $A \in \mathcal{U}$, then $\mathbb{N} \setminus A \notin \mathcal{U}$. Second, the intersection of sets in $\mathcal{U}$ must also be in $\mathcal{U}$. Third, finite sets must be excluded and cofinite sets (complements of finite sets) must be included.

We encode these constraints as a Boolean satisfiability (SAT) problem. Each subset of $\mathbb{N}$ that might be in the ultrafilter corresponds to a Boolean variable. The constraints become clauses in conjunctive normal form (CNF). For example, the requirement that exactly one of $L_{a,b}$, $E_{a,b}$, and $G_{a,b}$ is in the ultrafilter generates four clauses: one asserting that at least one is included, and three asserting that no two are simultaneously included.

When checking whether a set S must be in the ultrafilter, we add the clause $\neg S$ to our constraint system and check satisfiability. If the extended system is unsatisfiable, then S must be in the ultrafilter. Similarly, if adding S makes the system unsatisfiable, then S must be excluded. If both extensions are satisfiable, we have a choice point.

3.4 Fast Paths and Decidable Fragments

While SAT solving provides a complete decision procedure, many common comparisons can be resolved more efficiently. We identify several decidable fragments where membership can be determined directly:

Finite and Cofinite Sets Sets that are provably finite must be excluded from the ultrafilter, while cofinite sets must be included. For example, the set $\{n \in \mathbb{N} : n = 5\}$ is finite (containing only 5) and must be excluded. The set $\{n \in \mathbb{N} : n > 10\}$ is cofinite and must be included.

Comparisons with Constants When comparing expressions involving n and $1/n$ with constants, we can often determine the result directly. For instance, $n > 5$ generates the cofinite set $\{n \in \mathbb{N} : n > 5\}$, which must be in the ultrafilter. Therefore, the hyperreal n is greater than 5. Similarly, $1/n < 0.01$ generates a cofinite set and must hold.

Algebraic Simplification Before resorting to SAT solving, we perform algebraic simplification on sequence expressions. This simplification can reveal that sequences are constants or reduce complex expressions to simpler forms that fall into decidable fragments.

Series-Determined Order For expressions that admit a Laurent expansion in $\delta = 1/n$, we can often decide the eventual sign of $b_n - a_n$ by inspecting the dominant term of the expansion. If $b_n - a_n$ is eventually positive (resp. negative), then $L_{a,b}$ is cofinite (resp. finite), and the comparison is forced without SAT. This resolves pointwise-determined comparisons such as $(n+3) < (n+4)$ and analytic inequalities on infinitesimals such as $\sin(\delta) < \delta$.

3.5 The Choice Policy

When neither inclusion nor exclusion is forced by consistency, we must make a choice. Different choice policies lead to different ultrafilters and thus different hyperreal fields. Our default policy exhibits a "bias toward truth"—when forced to choose, we include sets rather than exclude them. This policy is arbitrary but consistent, ensuring crucial properties for practical computation.

Implications for Program Properties The deterministic nature of our choice policy has profound implications for program behavior. Reproducibility is guaranteed: a given program will always produce identical results across executions, which is essential for debugging and regression testing. This determinism contrasts with the mathematical non-determinism inherent in the ultrafilter selection, effectively canonicalizing one particular completion for each program execution pattern.

However, this determinism may introduce subtle biases. For instance, our bias toward inclusion means that underdetermined comparisons tend to evaluate to true, which could skew statistical properties in applications like Monte Carlo simulations. Programs that are sensitive to the distribution of truth values in underdetermined comparisons may exhibit unexpected behavior.

Alternative Policies Several alternative choice policies merit consideration. A randomized policy would select inclusion or exclusion with equal probability at each choice point, providing statistical sampling over the space of possible ultrafilters. This could be valuable for understanding the sensitivity of computations to ultrafilter choice. A user-guided policy could prompt for decisions at choice points, serving as a powerful debugging tool to understand how non-constructive choices affect program behavior. An adversarial policy could systematically explore worst-case behaviors, useful for verification and testing.

The choice policy becomes observable through program behavior. For instance, when comparing $\sin(n)$ with 0, no finite constraints determine the result. Our policy will choose to make $\sin(n) > 0$ if this comparison is encountered first. Subsequent related comparisons must respect this choice to maintain consistency, creating a path-dependent semantics where the order of comparisons can affect outcomes for underdetermined expressions.

4 Standard Part Extraction via Series Recognition

The (classical) standard part map st connects finite hyperreal computations back to real analysis. Our runtime exposes a *partial* extractor $\widehat{\mathrm{st}}$ implemented via series recognition. It returns a real number when it can certify (in a decidable fragment) that a hyperreal is near-standard with that standard part, and returns failure otherwise. Concretely, we recognize a subclass of values that admit a power series in $\delta = 1/n$ with no negative powers.

4.1 Series Representation

We call a hyperreal *series-near-standard* if it can be written as:

$$h = c_0 + c_1\delta + c_2\delta^2 + \cdots$$

where $c_i \in \mathbb{R}$ and $\delta = 1/n$. For such h, the (classical) standard part is $\mathrm{st}(h) = c_0$, and our extractor returns $\widehat{\mathrm{st}}(h) = c_0$. The presence of negative powers of δ (equivalently, positive powers of n) indicates that the hyperreal is infinite. By contrast, bounded oscillatory terms like $\sin(n)$ generally do not admit such an expansion and are handled by the ultrafilter machinery rather than the series recognizer.

Our series recognizer handles arithmetic operations and a collection of analytic functions. For arithmetic, we have:

$$(a_0 + a_1\delta + \cdots) + (b_0 + b_1\delta + \cdots) = (a_0 + b_0) + (a_1 + b_1)\delta + \cdots \tag{4}$$

$$(a_0 + a_1\delta + \cdots) \cdot (b_0 + b_1\delta + \cdots) = a_0b_0 + (a_0b_1 + a_1b_0)\delta + \cdots \tag{5}$$

Division requires special care. We support division only when the denominator is a monomial (a single power of δ). This restriction ensures that division doesn't introduce infinite series that our finite recognizer cannot handle.

4.2 Analytic Functions

For analytic functions, we compute series expansions using Taylor series. When evaluating $f(x_0 + \delta)$ where $x_0 \in \mathbb{R}$ and δ represents infinitesimal perturbations, we use:

$$f(x_0 + h) = \sum_{k=0}^{\infty} \frac{f^{(k)}(x_0)}{k!} h^k$$

where $h = c_1\delta + c_2\delta^2 + \cdots$ contains only positive powers of δ.

Our implementation supports the elementary functions sin, cos, tan, tanh, exp, $\log(1 + x)$, $\sqrt{1 + x}$, sinh, and cosh. For each function, we compute derivatives at the constant term and build the composition. The implementation truncates series at a configurable order (default 10), providing sufficient precision for most applications.

4.3 Limitations and Extensions

The series recognizer has inherent limitations. It cannot handle expressions with oscillatory terms like $\sin(n)$ or $(-1)^n$, as these do not admit series expansions. It also cannot handle general division where the denominator has multiple terms, as this would require computing infinite series inversions.

These limitations are fundamental rather than implementation restrictions. The hyperreal $\sin(n)$ is finite, but its classical standard part (its ultralimit) is completion-dependent: different ultrafilter completions can yield different real values. Consequently, no completion-invariant procedure can return a unique real number for $\sin(n)$ from finite runtime information, and our extractor returns failure on such terms. Similarly, expressions like $1/(\delta + \delta^2)$ would require infinite series to represent exactly.

Despite these limitations, the series recognizer successfully handles a significant fragment of hyperreal arithmetic, including all polynomial expressions in n and $1/n$, and compositions of analytic functions applied to near-zero arguments.

5 Implementation Architecture

Our implementation consists of four main components that work together to provide efficient hyperreal computation. The architecture separates concerns between mathematical representation, constraint management, satisfiability checking, and series manipulation.

5.1 Sequence Representation Layer

The foundation of our system is a domain-specific language for representing infinite sequences. Each sequence type implements a common interface providing evaluation at index n, algebraic simplification, and predicates for identifying infinitesimals and infinite sequences.

The base sequences include `Const(c)` for constant sequences, `NVar()` for the identity sequence n, `InvN()` for $1/n$, and `AltSign()` for $(-1)^n$. Composite sequences are built through operation nodes: `Add`, `Sub`, `Mul`, and `Div` for arithmetic, and `Sin`, `Cos`, `Tan`, `Tanh`, `Exp`, `Log1p`, `Sqrt1p`, `Sinh`, and `Cosh` for analytic functions.

Each sequence type implements simplification rules that exploit algebraic identities. For instance, `Mul(InvN(), NVar())` simplifies to `Const(1)`, and `Div(a, Const(c))` simplifies to `Mul(a, Const(1/c))` when $c \neq 0$. This simplification is crucial for performance, as it often reduces complex expressions to forms where fast paths apply.

5.2 Partial Ultrafilter Management

The `PartialUltrafilter` class maintains the growing collection of constraints that define our ultrafilter approximation. It tracks three types of information: the sets currently in the ultrafilter, the installed trichotomy partitions for compared pairs, and the mapping between sets and SAT variables.

When a comparison is requested, the partial ultrafilter first checks whether the result is determined by finite-or-cofinite analysis. Sets like $\{n : n < 10\}$ (finite) are immediately excluded, while sets like $\{n : n > 10\}$ (cofinite) are immediately included. This fast path avoids SAT solving for many common comparisons.

For comparisons that require SAT solving, the partial ultrafilter maintains incremental state. When a new set is included, we compute its intersections with all previously included sets. If any such intersection is provably finite, then the inclusion would violate non-principality; we therefore reject that branch (adding the appropriate blocking clause) and instead exclude the set. Otherwise, we add the intersections as committed-true sets, maintaining closure while keeping the representation finite.

5.3 SAT Solver Integration

Our SAT solver implements a standard DPLL algorithm with unit propagation. While more sophisticated SAT solvers exist, our implementation is sufficient because the constraint systems remain relatively small in practice.

The solver maintains statistics on the number of decisions, unit propagations, and conflicts. These statistics help us evaluate the effectiveness of our fast paths. In practice, most SAT calls require only unit propagation without any decision points, indicating that the constraints strongly determine the outcomes.

The CNF encoding includes several types of clauses. Complement clauses ensure that a set and its complement have opposite membership. Trichotomy clauses enforce that exactly one of $L_{a,b}$, $E_{a,b}$, and $G_{a,b}$ is in the ultrafilter. Transitivity clauses connect related comparisons: if $a < b$ and $b < c$, then $a < c$ must hold.

Opportunistic transitivity. For each compared pair (a, b) we install a trichotomy partition consisting of $L_{a,b}$, $E_{a,b}$, and $G_{a,b}$ together with exactly-one constraints (cf. Sec. 3). When partitions for (a, b) and (b, c) (resp. (c, a)) are already present, we opportunistically add the Horn clause

$$\neg L_{a,b} \lor \neg L_{b,c} \lor L_{a,c}$$

(resp. $\neg L_{c,a} \lor \neg L_{a,b} \lor L_{c,b}$). This prevents three-cycle inconsistencies while keeping the CNF compact by avoiding global $O(m^3)$ transitivity saturation.

5.4 Series Engine

The series engine performs Laurent series computations for standard part extraction. It represents series as dictionaries mapping integer exponents to real coefficients, supporting sparse representations efficiently.

The engine implements series arithmetic through coefficient manipulation. Addition combines coefficients of like powers, while multiplication convolves the series using the Cauchy product formula. For analytic functions, the engine computes Taylor series expansions around the constant term, then substitutes the infinitesimal parts.

The implementation carefully tracks precision throughout computations. Series are truncated at a configurable order (default 10) to keep computations finite and predictable, and represented sparsely as exponent-to-coefficient maps. These pragmatic choices balance precision with performance.

6 Correctness and Soundness

The correctness of our implementation rests on three pillars: the soundness of the partial ultrafilter construction, the validity of the series recognizer, and the consistency of the overall system. We establish each through a combination of theoretical analysis and empirical validation.

6.1 Partial Ultrafilter Soundness

Theorem 1 (Completion to a Non-Principal Ultrafilter)**.** *Let $\mathcal{U}_k$ be the family of committed-true index sets after k comparison operations, together with all cofinite subsets of $\mathbb{N}$. If this family contains no finite set and has the finite intersection property, then it extends to a non-principal ultrafilter on $\mathbb{N}$.*

This is a standard consequence of the ultrafilter lemma [9]: any proper filter extends to an ultrafilter, and containing the Fréchet filter (all cofinite sets) enforces non-principality. In the implementation, the SAT instance is a checkable proxy for consistency among the finitely many sets mentioned during execution, while our finite-or-cofinite fast paths explicitly exclude sets our analysis can prove finite and include sets it can prove cofinite.

Our implementation maintains several invariants that ensure soundness:

1. **Complement Consistency**: For every set S in our system, exactly one of S or $\mathbb{N} \setminus S$ is recorded as included or excluded.

2. **Intersection Closure (guarded)**: If $A, B \in \mathcal{U}_k$, then any non-principal completion must satisfy $A \cap B \in \mathcal{U}$ and therefore $A \cap B$ cannot be finite. Accordingly, when our analysis proves $A \cap B$ finite, we treat this as a conflict and forbid the joint inclusion by adding the blocking clause $\neg A \vee \neg B$, which forces at least one of the two sets to be excluded (operationally, we reject the tentative inclusion and commit the opposite choice). Otherwise, we commit $A \cap B \in \mathcal{U}_k$ (as unit-true), maintaining intersection closure.

3. **Finite/Cofinite Handling**: Finite sets are always excluded and cofinite sets are always included, consistent with non-principality.

4. **Trichotomy Preservation**: For each compared pair (a, b), exactly one of the three partition sets is included.

Proposition 1 (No three-cycles)**.** *With the opportunistic Horn clauses installed, it is inconsistent to have $L_{a,b}$, $L_{b,c}$, and $L_{c,a}$ simultaneously for any a, b, c.*

Proof. From $\neg L_{a,b} \vee \neg L_{b,c} \vee L_{a,c}$ and $\neg L_{c,a} \vee \neg L_{a,b} \vee L_{c,b}$, unit propagation on $L_{a,b}$ and $L_{b,c}$ forces $L_{a,c}$ and hence $\neg L_{c,a}$ by trichotomy, contradicting $L_{c,a}$. Rotations are symmetric. $\square$

6.2 Series Recognizer Validity

The series recognizer computes standard parts through Laurent series expansion. Its correctness depends on the mathematical validity of the series manipulations and the proper handling of convergence.

Proposition 2 (Series Recognition Soundness)**.** *If the series recognizer returns a standard part c for a hyperreal h, then $h - c$ is infinitesimal in any completion of the partial ultrafilter.*

This holds because our series computations respect the algebraic structure of hyperreals. The operations we support (arithmetic on series, composition with analytic functions) preserve the property of being near-standard. The truncation at finite order introduces only infinitesimal errors of higher order than the truncation point.

The recognizer correctly returns failure for expressions that are not near-standard. This includes expressions with negative powers of δ (infinite hyperreals) and expressions with oscillatory terms that prevent series expansion.

Regarding division, we distinguish between fundamental limitations and engineering choices. Division by a series with non-zero constant term is mathematically well-defined through geometric series expansion: $(1 + d_1\delta + d_2\delta^2 + \cdots)^{-1} = 1 - d_1\delta + (d_1^2 - d_2)\delta^2 + \cdots$. However, this expansion generally produces an infinite series that would require unbounded computation. Our restriction to division by monomials (single powers of δ) is therefore a pragmatic engineering decision that ensures all series representations remain finite and tractable within our truncated arithmetic framework. This choice balances mathematical generality with computational efficiency.

6.3 System Consistency

The overall system maintains consistency through careful interaction between components. When the partial ultrafilter makes a choice, it affects all future comparisons involving related expressions. The series recognizer operates independently of these choices for near-standard values, ensuring that standard parts are well-defined regardless of ultrafilter completion.

We validate consistency through a small executable validation suite, including:

1. algebraic and limit identities,
2. derivative checks via ϵ-perturbations, and
3. stress cases for choice-dependence and transitivity.

7 Foundational Perspective: Computing with Non-Constructive Objects

"Studying the monad, he understands the archangel."

—*The Kybalion* [19]

The epigraph is meant literally: we approach a large, non-constructive object through the smallest pieces that a computation can force us to confront. A terminating run cannot determine an entire ultrafilter, but it can determine a finite set of commitments about which index sets must be treated as "large." Our thesis is that this finite observational footprint—together with a classical completion theorem—is what entitles the program to speak as if it were executing inside some completed hyperreal field.

In this sense, the paper can be read as a precise operational counterpart to the Leibnizian practice of treating infinitesimals as fractions: the distinguished infinitesimal $1/n$ is literally the reciprocal of an infinite hyperinteger n (represented by the identity sequence), and, within the series-recognizable fragment, the extractor computes standard parts by isolating constant terms while treating higher-order terms as infinitesimal error.

Our implementation of hyperreals exemplifies a broader principle about computing with non-constructive mathematical objects. This section articulates the foundational framework that legitimizes such computation.

7.1 Finite Observation and Completion

Mathematical objects often exist only through non-constructive proofs, particularly those involving the Axiom of Choice. Non-principal ultrafilters are a canonical example: classical set theory proves they exist, but provides no explicit construction.

Our implementation does not attempt to construct an ultrafilter. Instead, it maintains a finite set of commitments about which index sets belong to the ultrafilter, represented as a CNF instance that we keep satisfiable. The key soundness statement is a completion theorem: when the committed sets avoid finiteness and satisfy the finite intersection property (e.g., by extending the cofinite filter), they extend to a non-principal ultrafilter. In this sense, the runtime constraint set is a finite observational prefix of some classical completion, sufficient for any terminating computation.

From a programming-language perspective, three ingredients make this viable:

Conservative Extension The non-constructive objects must extend a constructive base theory conservatively. For hyperreals, Łoś's theorem (transfer) ensures that all ultrafilter completions satisfy the same first-order theory of ordered fields, providing a stable semantic background.

Interface Abstraction Interaction with non-constructive objects must be mediated through a well-defined interface that does not reify the underlying completion. In our context, programs can only observe the answers to finitely many membership/comparison queries; the ultrafilter completion itself remains implicit.

Incremental Realization Finite computations must require only finite information about the non-constructive object. Our incremental construction commits only to finite portions of the ultrafilter, so a finite run requires only finite commitments. The complete ultrafilter need never be realized; satisfiability (together with the finiteness and finite-intersection hypotheses of the completion theorem) guarantees that the commitments made so far admit at least one classical completion.

7.2 Completion-Invariance of the Standard-Part Fragment

Different completions of a partial ultrafilter can disagree on order/equality tests for oscillatory sequences, so comparisons are inherently choice-dependent in our API. However, many computations in analysis do not branch on such comparisons; they compute standard parts of near-standard values. For this fragment, evaluation is invariant under ultrafilter completion.

Definition 2 (The $\widehat{\mathrm{st}}$–ring fragment)**.** *Let $\mathcal{L}_{\widehat{\mathrm{st}}}$ be the set of closed terms generated from real constants using $+, -, \cdot, /$ (where defined), the analytic constructors* $\sin, \cos, \tan, \tanh, \exp, \log(1 + \cdot), \sqrt{1+\cdot}, \cosh, \sinh$*, and the operator $\widehat{\mathrm{st}}$. An $\mathcal{L}_{\widehat{\mathrm{st}}}$ program evaluates these terms by interpreting the constructors pointwise on hyperreals and $\widehat{\mathrm{st}}(x)$ as $\mathrm{st}(x)$ when the extractor can certify that x is near-standard (otherwise it returns failure).*

Theorem 2 (Observational equivalence on $\mathcal{L}_{\widehat{\mathrm{st}}}$)**.** *For any two completions $\mathcal{U}$ and $\mathcal{U}'$ of the partial ultrafilter and any closed $\mathcal{L}_{\widehat{\mathrm{st}}}$ program P, the observable outcome of P (a real number or failure) is identical under $\mathcal{U}$ and under $\mathcal{U}'$.*

Proof sketch. In our implementation, the evaluation of all constructors in $\mathcal{L}_{\widehat{\mathrm{st}}}$ does not consult the ultrafilter, and $\widehat{\mathrm{st}}$ is computed by a local Laurent-series recognizer (returning $\mathrm{st}(x)$ when it succeeds). Therefore the result depends only on the underlying sequences, not on membership choices made by $\mathcal{U}$. See `Hyperreal.standard_part` (which calls `is_near_standard_by_series`) for the exact pathway. □

Remark 1 (Choice-dependence of comparisons)**.** *Outside $\mathcal{L}_{\widehat{\mathrm{st}}}$, order/equality tests on hyperreals (e.g. $x < y$) are observable and can be completion-dependent in our API. Branching on such comparisons* *may yield different outcomes across completions. This behavior is deliberate (cf. the* $(-1)^n$ *example in our demo).*

This theorem isolates a completion-invariant fragment: for programs that stay within $\mathcal{L}_{\widehat{\mathrm{st}}}$, observable outcomes do not depend on how the partial ultrafilter is completed. Choice-dependence becomes observable only when programs branch on comparisons such as $x < y$ for genuinely underdetermined sequences.

7.3 Relationship to Constructive Mathematics

Our approach differs fundamentally from constructive mathematics, which demands that existence proofs provide algorithms for construction. The existence of non-principal ultrafilters is established via the Axiom of Choice—a quintessentially non-constructive axiom that provides no construction algorithm.

Importantly, our system does not construct an ultrafilter. Instead, its correctness relies on the classical proof of ultrafilter existence, which guarantees that any consistent finite commitment set (meeting the hypotheses of our completion theorem) has a completion. Each choice made by the system is not a construction step but rather a commitment along one possible path in the vast, non-constructively-defined space of all ultrafilters.

Thus, we derive computational content from classical, non-constructive proofs without making the mathematics constructive. We make non-constructive mathematics computable by showing how to interact with objects whose existence is merely guaranteed, lazily resolving their properties as needed. This subtle but profound distinction differentiates our approach from both constructive frameworks like those of Schmieden and Laugwitz [17] and Palmgren [15], and from axiomatic approaches like Nelson's IST [13].

This perspective suggests a middle path between classical and constructive mathematics for computational purposes. We need not restrict ourselves to constructive objects, nor must we despair at the non-constructive nature of classical mathematics. Instead, we can build sound computational interfaces to classical structures through appropriate abstraction boundaries.

The success of this approach with hyperreals suggests a general recipe for other non-constructive objects: expose an interface that makes only finitely many queries in any finite run, maintain a finite set of commitments, and rely on a classical completion theorem to justify soundness.

8 Evaluation

We evaluate our implementation across three dimensions: the effectiveness of fast paths in avoiding SAT solving, the growth rate of the partial ultrafilter, and the coverage of the series recognizer. Our results demonstrate that the lazy approach is not only theoretically sound but also practically efficient.

8.1 Experimental Setup

Our evaluation uses seven benchmark tasks that exercise different aspects of the system:

1. **Near-Standard Limits**: Computing derivatives and limits using infinitesimals
2. **Finite vs. Cofinite**: Testing fast paths for finite and cofinite set detection
3. **Alternating Sign**: Handling the choice-dependent sequence $(-1)^n$

Task	Queries	Fast	Cache	SAT Calls	Commits	Size Δ	Solves	Dec.
Near-Standard Limits	0	0	0	0	0	0	0	0
Finite vs. Cofinite	5	5	0	0	0	2	0	0
Alternating Sign	3	1	0	4	1	1	5	5
Sine Grid	12	0	0	24	6	6	30	30
Cross-Links	3	2	1	0	0	0	0	0
Intersection Growth	12	2	3	14	7	9	21	28
Equality Filters	5	5	0	0	0	0	0	0

Table 1: Evaluation metrics for hyperreal operations. Tasks exercise cofinite/finite fast paths, trichotomy conflicts, and intersection-closure growth.

4. **Sine Grid**: Determining signs of $\sin(n + k)$ for multiple offsets
5. **Cross-Links**: Testing consistency maintenance across related comparisons
6. **Intersection Growth**: Measuring closure computation overhead
7. **Equality Filters**: Testing finite set exclusion through equality comparisons

We measure several metrics for each task: the number of SAT solver invocations, the size of the partial ultrafilter (committed sets), the number of intersection closures computed, and the number of unit propagations and decisions in SAT solving. The detailed results are presented in Table 1.

8.2 Fast Path Effectiveness

Our fast paths successfully avoid SAT solving for a meaningful fraction of comparisons. Across our benchmark suite (40 membership queries), 19 queries (47.5%) are resolved without SAT: 15 are discharged through finite-or-cofinite reasoning, and 4 are answered by reusing previously committed sets (including repeated queries and direction-swap invariance). The remaining comparisons predominantly involve oscillatory sequences such as $\sin(n + k)$ that necessarily defer to choice-dependent reasoning.

The finite-or-cofinite detection is particularly effective. All comparisons of the form $n < c$ or $n > c$ for constant c are resolved immediately, as are comparisons like $1/n < \epsilon$ for positive ϵ. These patterns appear frequently in limit and derivative computations, explaining the high fast-path hit rate.

8.3 Partial Ultrafilter Growth

The partial ultrafilter grows moderately even for complex computations. Starting from the baseline commitment to $\mathbb{N}$, our benchmark suite increases the number of committed (included) sets to at most 19. The largest jump (9 additional commitments) arises in the intentionally adversarial "intersection growth" scenario; typical tasks stay at or below 10 committed sets, indicating that many comparisons reuse existing constraints rather than adding new ones.

The intersection closure adds overhead, as each newly included set must be intersected with all previously included sets. However, the total number of sets remains manageable due to our incremental approach. We only compute closures for sets that are actually included, not for all possible intersections.

8.4 SAT Solver Performance

When SAT solving is required, the workload is skewed. Our benchmarks perform 56 SAT solves: 42 disambiguation calls from membership queries, plus 14 additional finite-intersection-property checks when validating commitments. Across all runs, the solver performs 9,975 unit propagations and 63 branching decisions; the intersection-growth stress test accounts for 6,650 propagations and 28 decisions across 21 solves.

The CNF formula grows gradually as comparisons are made. Transitivity constraints are added lazily only when comparison triangles close, keeping the formula size manageable. In our benchmarks, formulas range from 6 clauses at initialization to 821 clauses at peak; typical tasks stay below about 300 clauses, while the intersection-growth case reaches 774 clauses. Even this worst case stays within the capabilities of our lightweight DPLL implementation.

8.5 Series Recognizer Coverage

The series recognizer successfully computes standard parts for all near-standard expressions in our benchmarks. It correctly identifies infinite and oscillatory expressions, returning failure appropriately. The truncation at order 10 provides sufficient precision for our test cases, achieving errors between 10^{-9} and 10^{-12} for standard part extraction.

The recognizer's performance is predictable: arithmetic operations have linear complexity in the series order, while function compositions have quadratic complexity due to series multiplication. For our default order of 10, all operations complete in microseconds on modern hardware.

8.6 Performance Boundaries and Limitations

Our implementation is a research prototype with several limitations. The series recognizer operates in floating-point and uses finite truncation; division support is intentionally restricted to keep series representations tractable. The SAT component is a lightweight DPLL solver intended for incremental consistency checks rather than large industrial CNF instances.

While our evaluation demonstrates strong performance on typical use cases, we acknowledge that certain adversarial patterns could stress the system. Highly interdependent oscillating sequences, such as simultaneous comparisons between $\sin(n)$, $\sin(2n)$, and $\sin(n^2)$, would generate complex webs of constraints that could challenge the SAT solver. In such cases, the constraint graph becomes densely connected, potentially requiring extensive search beyond unit propagation.

Our current benchmarks focus on demonstrating the system's effectiveness for common computational patterns rather than exploring worst-case scenarios. The absence of adversarial test cases in our evaluation represents a deliberate choice to validate the practical utility of the approach rather than to characterize its theoretical limits. Future work should explore these boundaries more systematically, particularly for applications requiring extensive comparisons between non-convergent sequences.

The system's performance degrades gracefully as constraint complexity increases. When the fast paths cannot resolve comparisons and the SAT solver requires multiple decision points, computation time increases but correctness is preserved. The partial ultrafilter's growth remains bounded by the number of distinct comparisons, ensuring that memory usage scales linearly with program complexity rather than with the theoretical size of the complete ultrafilter.

8.7 Comparison with Automatic Differentiation

We compared our hyperreal-based derivative computation with a standard automatic differentiation implementation. For smooth functions, both approaches yield identical results to machine precision. On a microbenchmark that differentiates x^3 at $x = 2$ for 1000 iterations, the hyperreal approach is roughly 200× slower than forward-mode dual numbers (0.096 s vs. 0.00046 s) because it maintains symbolic sequence objects, partial ultrafilter state, and SAT metadata alongside the numeric computation.

However, our goal differs from forward-mode dual numbers. For analytic expressions, our series recognizer is closely related to Taylor-mode AD (truncated power series): evaluating $f(x + \epsilon)$ encodes higher derivatives in the coefficients of ϵ^k up to the truncation order. The distinctive contribution here is integrating that higher-order series view with hyperreal comparison semantics backed by a lazy ultrafilter, so computations can also expose completion-dependence and detect non-differentiability by probing multiple infinitesimal directions.

To illustrate the distinction, consider the function $f(x) = |x|$ at $x = 0$. Standard forward-mode AD, following the execution trace, computes a one-sided derivative depending on the branch taken, but does not expose the non-differentiability. In nonstandard analysis, the derivative exists only if $(f(h) - f(0))/h$ has the same standard part for *all* nonzero infinitesimals h. In our implementation, $h = 1/n$ is a positive infinitesimal and yields the right derivative; to detect non-differentiability one must also compare against $-1/n$ (or use a sign-indeterminate infinitesimal such as $(-1)^n/n$), in which case the quotient becomes completion-dependent: its classical standard part exists but can vary across completions (e.g., $\mathrm{st}((-1)^n) = \pm 1$). Our extractor therefore returns failure in this case. This principled handling of discontinuities justifies the performance overhead for applications requiring mathematical rigor at non-smooth points.

9 Related Work

9.1 Nonstandard Analysis in Computer Science

The application of nonstandard analysis to computer science has a long history, with several distinct approaches emerging over the decades. Schmieden and Laugwitz [17] pioneered an early constructive approach using sequences of rationals, providing a computational foundation for infinitesimals before Robinson's model-theoretic framework. Their work presaged many ideas we employ, though without the ultrafilter machinery that ensures consistency.

Nelson's Internal Set Theory (IST) [13] offers an axiomatic approach to nonstandard analysis that avoids explicit ultrafilter constructions by extending ZFC with new axioms governing standard and nonstandard objects. While IST provides an elegant foundation for reasoning about infinitesimals, it does not directly address computational realization. Our work can be viewed as providing an operational semantics for a fragment of IST, where the standard/nonstandard distinction emerges from our partial ultrafilter construction.

These alternative foundations highlight different paths to infinitesimals. Schmieden-Laugwitz provides constructive sequences, Nelson offers axiomatic extensions, and Robinson employs model theory. Our contribution synthesizes insights from these approaches: we use Robinson's ultrafilter framework for soundness, embrace computational aspects like Schmieden-Laugwitz, and achieve the accessibility that Nelson sought through axiomatization, but through lazy computation rather than new axioms.

Keisler [10] introduced elementary calculus using infinitesimals, demonstrating their pedagogical value. Our work realizes this approach computationally, enabling students and practitioners to compute with the same infinitesimal concepts that Keisler advocated for mathematical education.

Several researchers have explored nonstandard models in program semantics and hybrid systems. Benveniste et al. [3] propose a non-standard semantics of hybrid-system modelers, while Hasuo and Suenaga [7] apply nonstandard analysis to static analysis of hybrid systems. These works use nonstandard analysis as a mathematical tool but do not implement computational infinitesimals.

In theorem proving, Fleuriot [5] formalized nonstandard analysis in Isabelle/HOL, providing machine-checked proofs using hyperreals. Benl et al. [2] developed a constructive approach using Cauchy sequences. Our work differs in focusing on executable implementations rather than formal verification.

9.2 Computational Approaches to Infinitesimals

Previous computational approaches to infinitesimals have taken different paths from our ultrafilter-based method. Automatic differentiation [6, 1] computes derivatives via dual numbers, computation graphs, or truncated power series (Taylor-mode). For analytic expressions, our series recognizer aligns closely with the Taylor-mode viewpoint; our distinguishing feature is pairing that capability with an ultrafilter-backed hyperreal semantics for order/equality, where comparisons can be completion-dependent and are enforced consistently by the lazy partial ultrafilter.

Computer algebra systems like Mathematica and Maple support symbolic reasoning about limits, infinite values, and related indeterminate forms (and, in some approaches, infinitesimals) [4]. These systems work symbolically rather than providing a numeric model of hyperreals. Our approach bridges symbolic and numeric computation through the partial ultrafilter construction.

The field of constructive nonstandard analysis [15] seeks to develop infinitesimals without the Axiom of Choice. While philosophically appealing, this approach has not yielded practical implementations. Our work embraces classical foundations while achieving computational tractability.

9.3 SAT Solving and Constraint Systems

Our use of SAT solving for maintaining ultrafilter consistency connects to broader work on constraint programming and automated reasoning. The architecture of our system, where a primary algorithm makes queries to a solver that incrementally adds constraints, follows the lazy SMT paradigm established by Sebastiani [18]. In lazy SMT, a SAT engine explores Boolean structure while theory-specific solvers check consistency and generate explanatory clauses. Our system can be viewed as implementing SMT for the "theory of ultrafilters," where the partial ultrafilter manager acts as a theory solver generating ultrafilter-specific constraints.

Similarly, the technique of lazy clause generation [14] in constraint programming operates on the same principle, with constraint propagators lazily generating clauses to explain their deductions. Our incremental construction of trichotomy and transitivity constraints follows this pattern, adding clauses only as needed rather than eagerly encoding all possible constraints.

The encoding of mathematical structures as SAT problems has been successful in other domains. The Boolean Pythagorean triples problem [8] and the resolution of Erdős discrepancy conjecture [11] demonstrate SAT solving's power for mathematical questions. Our encoding is simpler but must maintain consistency across an evolving constraint system, making the lazy approach essential for scalability.

10 Future Directions

Our implementation opens several avenues for future research, both in extending the current system and in applying the underlying principles to other domains.

10.1 Extended Function Support

The current series recognizer handles a limited set of analytic functions. Extending support to special functions (Bessel functions, elliptic integrals), algebraic functions (through Puiseux series), and implicitly defined functions would broaden the system's applicability. Each extension requires careful analysis of series convergence and composition properties.

10.2 Alternative Choice Policies

Our current implementation uses a fixed choice policy when the partial ultrafilter allows freedom. Investigating alternative policies could yield interesting properties. A randomized policy would provide probabilistic guarantees about limiting behavior. An adversarial policy could find corner cases in hyperreal algorithms. Learning-based policies could adapt to typical usage patterns.

10.3 Parallel Exploration

The lazy construction naturally suggests parallel exploration of different ultrafilter completions. By maintaining multiple partial ultrafilters in parallel, we could explore the space of possible behaviors simultaneously. This would be particularly valuable for understanding the sensitivity of computations to ultrafilter choice.

10.4 Applications to Other Non-Constructive Objects

The completion-based recipe suggested by our ultrafilter interface may apply to other non-constructive mathematical objects. Prime candidates include non-measurable sets (accessed through an appropriate notion of integration), non-computable reals (accessed through approximation schemes), and generic filters in forcing (accessed through truth values of forced statements). Each application would require identifying an interface whose finite observations admit a suitable completion theorem.

10.5 Integration with Proof Assistants

Connecting our implementation to proof assistants would enable verified computations with hyperreals. The partial ultrafilter construction could generate proof certificates that witness the consistency of choices made during execution. This would provide the best of both worlds: efficient computation with formal correctness guarantees.

11 Conclusion

This paper has demonstrated that hyperreal numbers, despite their non-constructive foundations, can be implemented efficiently for practical computation. Our lazy partial ultrafilter construction resolves the apparent paradox of computing with objects that cannot be explicitly constructed, while our fast paths and series recognizer provide practical efficiency.

The completion-based perspective extends beyond hyperreals to suggest a general approach for computing with non-constructive mathematical objects: maintain a finite, checkable prefix of commitments and rely on a classical completion theorem to justify soundness with respect to some full model.

Our implementation successfully handles diverse hyperreal computations, from limit evaluation to derivative calculation, while maintaining mathematical rigor. The system's efficiency, demonstrated through extensive evaluation, validates our design choices and shows that the overhead of managing partial ultrafilters is acceptable for many applications.

Looking forward, this work opens new possibilities at the intersection of nonstandard analysis and computer science. Hyperreals provide an elegant framework for reasoning about limits, continuity, and approximation—concepts central to both mathematical analysis and program semantics. By making hyperreals computational, we enable their application to practical problems in scientific computing, program analysis, and beyond.

The success of our approach suggests that the computational content of classical mathematics is richer than commonly believed. Non-constructive objects need not be purely theoretical. With appropriate abstraction boundaries, they can support practical computation. This perspective offers a path forward for implementing other mathematical structures whose existence relies on non-constructive principles, potentially unlocking new computational capabilities across mathematics and computer science.

Data Availability

The implementation and evaluation benchmarks are available at github.com/krzysztofwos/hyperreals. The artifact includes the complete source code and the evaluation/validation scripts needed to reproduce all experimental results reported in this paper.

Acknowledgements

I thank Stephen Fitz for debates on the ontological status of the continuum. His insistence that bounded agents can only meaningfully manipulate discrete symbols—and that continuous mathematics involves "words that don't actually mean anything"—motivated me to demonstrate precisely how finite symbol manipulation can cohere with non-constructive structures without requiring their completion.

References

[1] Atılım Güneş Baydin, Barak A. Pearlmutter, Alexey Andreyevich Radul, and Jeffrey Mark Siskind. Automatic differentiation in machine learning: a survey. *Journal of Machine Learning Research*, 18(153):1–43, 2018. URL http://jmlr.org/papers/v18/17-468.html.

[2] Holger Benl, Ulrich Berger, Helmut Schwichtenberg, Monika Seisenberger, and Wolfgang Zuber. Proof theory at work: Program development in the Minlog system. In Wolfgang Bibel and Peter H. Schmitt, editors, *Automated Deduction – A Basis for Applications*, volume 9 of *Applied Logic Series*, pages 41–71. Kluwer Academic Publishers, Dordrecht, 1998. doi: 10.1007/978-94-017-0435-9_2. URL https://doi.org/10.1007/978-94-017-0435-9_2. Chapter in Volume II: Systems and Implementation Techniques.

[3] Albert Benveniste, Timothy Bourke, Benoît Caillaud, and Marc Pouzet. Non-standard semantics of hybrid systems modelers. *Journal of Computer and System Sciences*, 78(3):877–910, 2012.

doi: 10.1016/j.jcss.2011.08.009. URL https://www.sciencedirect.com/science/article/pii/S0022000011001061.

[4] Richard J. Fateman. To infinity and beyond. Manuscript, 2022. URL https://people.eecs.berkeley.edu/~fateman/papers/infinity.pdf.

[5] Jacques Désiré Fleuriot. A combination of geometry theorem proving and nonstandard analysis, with application to Newton's Principia. Technical Report UCAM-CL-TR-469, University of Cambridge, Computer Laboratory, August 1999. URL https://www.cl.cam.ac.uk/techreports/UCAM-CL-TR-469.pdf. Technical report based on a dissertation submitted March 1999 for the degree of PhD.

[6] Andreas Griewank and Andrea Walther. *Evaluating Derivatives: Principles and Techniques of Algorithmic Differentiation*. SIAM, 2nd edition, 2008. doi: 10.1137/1.9780898717761.

[7] Ichiro Hasuo and Kohei Suenaga. Exercises in nonstandard static analysis of hybrid systems. In P. Madhusudan and Sanjit A. Seshia, editors, *Computer Aided Verification – CAV 2012*, volume 7358 of *Lecture Notes in Computer Science*, pages 462–478. Springer, 2012. doi: 10.1007/978-3-642-31424-7_34. URL https://doi.org/10.1007/978-3-642-31424-7_34. Extended version with proofs available at https://group-mmm.org/ ichiro/papers/cav2012Extended.pdf.

[8] Marijn J. H. Heule, Oliver Kullmann, and Victor W. Marek. Solving and verifying the Boolean Pythagorean triples problem via cube-and-conquer. In Nadia Creignou and Daniel Le Berre, editors, *Theory and Applications of Satisfiability Testing – SAT 2016*, volume 9710 of *Lecture Notes in Computer Science*, pages 228–245. Springer, 2016. doi: 10.1007/978-3-319-40970-2_15.

[9] Thomas Jech. *Set Theory: The Third Millennium Edition, Revised and Expanded*. Springer Monographs in Mathematics. Springer, 2003. doi: 10.1007/3-540-44761-X.

[10] H. Jerome Keisler. *Foundations of Infinitesimal Calculus*. Prindle, Weber & Schmidt, 1976.

[11] Boris Konev and Alexei Lisitsa. A SAT attack on the Erdős discrepancy conjecture. In Carsten Sinz and Uwe Egly, editors, *Theory and Applications of Satisfiability Testing – SAT 2014*, volume 8561 of *Lecture Notes in Computer Science*, pages 219–226. Springer, 2014. doi: 10.1007/978-3-319-09284-3_17.

[12] Jerzy Łoś. Quelques remarques, théorèmes et problèmes sur les classes définissables d'algèbres. In *Mathematical Interpretation of Formal Systems*, pages 98–113. North-Holland, 1955. doi: 10.1016/S0049-237X(09)70306-4.

[13] Edward Nelson. Internal set theory: a new approach to nonstandard analysis. *Bulletin of the American Mathematical Society*, 83(6):1165–1198, 1977. doi: 10.1090/S0002-9904-1977-14398-X.

[14] Olga Ohrimenko, Peter J. Stuckey, and Michael Codish. Propagation = lazy clause generation. In *Principles and Practice of Constraint Programming – CP 2007*, volume 4741 of *Lecture Notes in Computer Science*, pages 544–558. Springer, 2007. doi: 10.1007/978-3-540-74970-7_39. URL https://doi.org/10.1007/978-3-540-74970-7_39.

[15] Erik Palmgren. Developments in constructive nonstandard analysis. *Bulletin of Symbolic Logic*, 4(3):233–272, 1998. doi: 10.2307/421031.

[16] Abraham Robinson. *Non-Standard Analysis*. North-Holland, 1966.

[17] Curt Schmieden and Detlef Laugwitz. Eine erweiterung der infinitesimalrechnung. *Mathematische Zeitschrift*, 69:1–39, 1958. doi: 10.1007/BF01187391. In German. English translation: “An Extension of Infinitesimal Calculus”.

[18] Roberto Sebastiani. Lazy satisfiability modulo theories. *Journal on Satisfiability, Boolean Modeling and Computation*, 3(3-4):141–224, 2007. doi: 10.3233/SAT190034. URL https://doi.org/10.3233/SAT190034.

[19] Three Initiates. *The Kybalion: A Study of the Hermetic Philosophy of Ancient Egypt and Greece*. Yogi Publication Society, Chicago, 1908.