Computational Basis Transforms: A Unified Framework for Domain Transformation Algorithms

Alexander Towell
Department of Computer Science
Southern Illinois University Edwardsville
atowell@siue.edu

September 16, 2025

Abstract

What if computational complexity were not inherent to problems, but merely artifacts of our choice of representation? We present Computational Basis Transforms (CBT), a unifying framework revealing that diverse algorithmic breakthroughs—from FFT to automatic differentiation—share a common pattern: transforming to domains where hard operations become easy. We formalize this insight mathematically, proving a fundamental No Free Lunch theorem that quantifies the inevitable trade-offs in any domain transformation. Through analysis of ten concrete transforms, we demonstrate 8–70× speedups for domain-appropriate operations while enabling previously infeasible computations (e.g., stable products of millions of near-zero probabilities). Beyond individual transforms, we identify composition patterns yielding emergent efficiencies and establish direct inter-domain mappings that preserve numerical stability. Our open-source C++17 implementation makes these theoretical insights immediately practical. By showing that efficiency depends on representation choice rather than problem structure, CBT suggests a paradigm shift: don't optimize algorithms within domains—transform to domains where your operations are naturally efficient. This work provides both rigorous theory for understanding computational trade-offs and practical tools for exploiting them.

1 Introduction

Many of the most significant algorithmic advances in computing share a unifying but underappreciated pattern: they achieve computational efficiency not by optimizing operations within a fixed representation, but by transforming problems into alternative domains where the desired operations have fundamentally different complexity characteristics. The Fast Fourier Transform [8] reduces convolution from $O(n^2)$ to $O(n \log n)$ by exploiting the frequency domain's algebraic structure. Logarithmic arithmetic [21] transforms multiplication—a complex operation in positional number systems—into addition, a simpler operation that also prevents catastrophic underflow in floating-point computation. Quaternions [13] eliminate the singularities inherent in Euler angle representations by embedding 3D rotations in a 4D algebraic structure.

Despite their widespread use and profound impact, these techniques have been studied primarily in isolation within their respective application domains. The FFT is treated as a signal processing tool, logarithmic arithmetic as a numerical stability technique, and quaternions as a computer graphics primitive. This compartmentalization has obscured a deeper principle: these are all instances of a general computational strategy that we formalize as *Computational Basis Transforms* (CBT).

This paper presents CBT as a unifying theoretical framework that reveals the common structure underlying these disparate techniques. Building on foundations from category theory [2], type theory [4], and numerical analysis [14], we provide rigorous definitions, prove fundamental limits, and demonstrate practical applications. Our key insight is that computational complexity is not inherent to problems but rather depends on the choice of representation—and systematic transformation between representations can yield dramatic efficiency gains.

1.1 Motivating Examples

Consider these seemingly unrelated computational challenges:

1. **Extreme Dynamic Range**: Computing $\prod_{i=1}^{10^6} p_i$ where each $p_i \approx 10^{-10}$ causes underflow after just 30 terms in standard floating-point.

2. **Bayesian Inference**: Sequential probability updates require expensive normalization after each step.

3. **Parallel Arithmetic**: Addition with carry propagation is inherently sequential, limiting parallelization.

4. **Exact Rational Computation**: Floating-point arithmetic accumulates errors in iterative algorithms.

Each problem becomes tractable through domain transformation:

```
// Problem 1: Logarithmic transform prevents underflow
cbt::lg<double> product = cbt::lg<double>::from_value(p[0]);
for(size_t i = 1; i < n; ++i) {
    product = product * cbt::lg<double>::from_value(p[i]);
}  // Stable for millions of small probabilities

// Problem 2: Odds-ratio eliminates normalization
cbt::odds_ratio<double> posterior = prior;
posterior = posterior * likelihood_ratio;  // O(1) update

// Problem 3: RNS enables parallel arithmetic
cbt::rns<int64_t, 251, 253, 255> a = cbt::rns<>::from_integer(x
    );
cbt::rns<int64_t, 251, 253, 255> b = cbt::rns<>::from_integer(y
    );
auto sum = a + b;  // Each modulus computed independently

// Problem 4: Stern-Brocot provides exact rationals
cbt::stern_brocot r = cbt::stern_brocot::approximate_real(M_PI,
     1e-6);
auto squared = r * r;  // Exact rational result
```

Listing 1: Solutions through CBTs

1.2 CBT Selection Guide

Table 1 provides a decision framework for selecting appropriate CBTs based on your computational requirements:

Decision Criteria:

1. **Identify dominant operations**: What operations occur most frequently in your workload?

2. **Assess trade-offs**: Can you afford the degraded operations for the benefit gained?

3. **Consider composition**: Can multiple CBTs be combined for emergent benefits?

4. **Evaluate overhead**: Is the transformation cost amortized over enough operations?

Table 1: CBT Decision Framework: Selecting the Right Transform for Your Problem

If You Need...	**Use CBT**	**Benefits**	**Trade-offs**
Stable products of many small probabilities	Logarithmic Logarithmic	Multiplication $\rightarrow$ addition: $O(1)$	Addition becomes complex: $O(n)$
Fast Bayesian updates without normalization	Odds-Ratio Odds-Ratio	No normalization needed: $O(1)$	Direct probability access costly
Exact rational arithmetic for iterative algorithms	Stern-Brocot Stern-Brocot	No rounding errors, exact results	Limited to rationals, tree navigation overhead
Parallel arithmetic without carries	Residue Number System (RNS)	No carry propagation, fully parallel	Division complex, magnitude comparison hard
Handle 200+ orders of magnitude	Multiscale Multiscale	Extreme dynamic range: 10^{-300} to 10^{300}	Storage overhead, complex operations
Automatic derivatives in one pass	Dual Numbers Dual Numbers	Exact derivatives computed alongside	Double storage, higher-order costly
Guaranteed error bounds for numerical computation	Interval Arithmetic	Rigorous bounds on all operations	Conservative estimates, dependency problem
Optimization problems with max/min operations	Tropical Tropical	Max-plus algebra, shortest paths	Limited algebraic structure
3D rotations without gimbal lock	Quaternions Quaternions	No gimbal lock, smooth interpolation	4D representation, non-intuitive
Cryptographic operations with large primes	Modular Modular	Efficient modular exponentiation	Limited to modular arithmetic only

1.3 Contributions

This paper makes four key contributions:

1. **Unified Theory**: We develop a rigorous mathematical framework that reveals the common structure underlying diverse algorithmic techniques—from FFT to automatic differentiation—as instances of domain transformation. We prove a fundamental No Free Lunch theorem establishing that computational trade-offs are inevitable.

2. **Systematic Analysis**: We analyze ten transforms within our framework, uncovering novel connections like direct logarithmic-to-multiscale mappings that avoid numerical instability. We provide decision criteria for transform selection based on workload characteristics.

3. **Practical Impact**: Our empirical evaluation demonstrates 8–70× speedups for appropriate operations while enabling previously infeasible computations. We provide memory overhead analysis, failure cases, and comparison with specialized libraries.

4. **Emergent Patterns**: We identify composition patterns that yield efficiencies beyond individual transforms and establish conditions for information-preserving inter-domain mappings, with immediate applications in numerical computing.

1.4 Organization

Section 2 presents formal definitions and theoretical foundations. Section 3 analyzes specific CBT instances. Section 4 explores inter-domain transformations. Section 5 discusses CBT composition. Section 6 demonstrates applications across computing domains. Section 7 provides empirical evaluation. Section 8 surveys related work. Section 9 concludes with limitations and future directions.

2 Formal Framework

2.1 Intuition

Before diving into formal definitions, let's build intuition through a simple example that we'll thread throughout this section.

Example 1 (The Power of Representation)**.** Consider computing $2^{100} \times 2^{100}$. In standard decimal representation, this requires multiplying two 31-digit numbers—a complex operation. But if we transform to a logarithmic representation where we store the exponent instead of the value:

- Standard domain: $2^{100} = 1267650600228229401496703205376$ (31 digits)
- Logarithmic domain: Store 100 (the exponent)
- Multiplication becomes: $100 + 100 = 200$ (simple addition)
- Result: 2^{200} (represented as 200)

This transformation trades multiplication complexity for addition simplicity. However, there's no free lunch—other operations become harder. Adding $2^{100} + 2^{100}$ in logarithmic domain requires computing $\log_2(2 \cdot 2^{100}) = 101$, which involves converting back to standard representation.

This example illustrates the key insight of CBTs: *computational complexity depends on representation.* Different representations make different operations efficient. The art lies in choosing the right representation for your workload.

2.2 Basic Definitions

We formalize computational domains and their transformations using category-theoretic structures [2].

Definition 1 (Computational Domain)**.** A **computational domain** is a tuple $D = (S_D, O_D, P_D, \mathcal{C}_D)$ where:

- S_D is the state space (the set of representable values)
- $O_D = \bigcup_{n \geq 0} O_D^{(n)}$ where $O_D^{(n)} \subseteq \{f : S_D^n \to S_D\}$ denotes the set of n-ary operations
- $P_D \subseteq \{p : S_D \to \{\text{true}, \text{false}\}\}$ is the set of decidable predicates
- $\mathcal{C}_D : O_D \times \mathbb{N} \to \mathbb{R}_{\geq 0}$ assigns computational complexity (in appropriate units) to each operation $\omega \in O_D$ for input size $n \in \mathbb{N}$

We require that O_D includes at least the identity function $\text{id} : S_D \to S_D$ and that $\mathcal{C}_D$ satisfies reasonable axioms (monotonicity, sub-additivity for composition).

Example 2. The standard floating-point domain $D_{\mathbb{R}}$ has $S_{D_{\mathbb{R}}} = \{x \in \mathbb{R} : |x| \in [0, 2^{1024}]\}$ (IEEE 754 double), $O_{D_{\mathbb{R}}}$ includes $\{+, -, \times, \div, \sqrt{}, \ldots\}$, and $\mathcal{C}_{D_{\mathbb{R}}}(\times, n) = O(1)$ for hardware multiplication.

Definition 2 (Computational Basis Transform)**.** A **Computational Basis Transform (CBT)** is a tuple $(D, D', \phi, \phi^{-1}, \Omega)$ where:

- $D = (S_D, O_D, P_D, \mathcal{C}_D)$ is the source domain
- $D' = (S_{D'}, O_{D'}, P_{D'}, \mathcal{C}_{D'})$ is the target domain
- $\phi : S_D \to S_{D'}$ is an injective transform function
- $\phi^{-1} : \text{Im}(\phi) \to S_D$ is the inverse transform, where $\text{Im}(\phi) = \{\phi(x) : x \in S_D\} \subseteq S_{D'}$
- $\Omega = (\text{Improved}, \text{Degraded}, \text{Cost})$ quantifies the computational trade-offs:

- Improved $\subseteq O_D$: operations that become more efficient in D'
- Degraded $\subseteq O_D$: operations that become less efficient in D'
- Cost : $\mathbb{N} \to \mathbb{R}_{\geq 0}$: transformation overhead as a function of input size

where $\phi_*(\omega)$ denotes the induced operation in D' that preserves semantics (when such exists).

Definition 3 (Homomorphic Property)**.** A CBT $\phi : D \to D'$ is **homomorphic** with respect to operation $\omega \in O_D^{(n)}$ if there exists $\omega' \in O_{D'}^{(n)}$ such that:

$$\forall (x_1, \ldots, x_n) \in S_D^n : \phi(\omega(x_1, \ldots, x_n)) = \omega'(\phi(x_1), \ldots, \phi(x_n)) \tag{1}$$

When this holds, we say ω' is the **image** of ω under ϕ, denoted $\omega' = \phi_*(\omega)$.

Remark 1. The homomorphic property ensures that computations can be performed entirely in the transformed domain without loss of correctness. This is the key enabler of computational advantages in CBTs.

2.3 The No Free Lunch Theorem

Theorem 1 (No Free Lunch for CBTs)**.** Let $(D, D', \phi, \phi^{-1}, \Omega)$ be a non-trivial CBT where $\phi : S_D \to S_{D'}$ is not merely a relabeling (i.e., not just a bijection with $\mathcal{C}_{D'}(\omega', n) = \mathcal{C}_D(\omega, n)$ for all corresponding operations). Then:

$$\Omega_+ \neq \emptyset \implies \Omega_- \neq \emptyset \text{ or } \Omega_c(n) > 0 \text{ for some } n \in \mathbb{N} \tag{2}$$

In other words, every CBT that improves the complexity of some operations must either:

(i) Degrade the complexity of other operations ($\Omega_- \neq \emptyset$), or

(ii) Incur non-zero transformation overhead ($\exists n : \Omega_c(n) > 0$), or

(iii) Both.

Proof. We provide three complementary arguments that together establish the theorem.

Part 1: Information-Theoretic Argument. Let $(D, D', \phi, \phi^{-1}, \Omega)$ be a CBT with $\Omega_+ \neq \emptyset$. Since ϕ is injective, it preserves information content. For any random variable X over S_D, the Shannon entropy satisfies:

$$H(\phi(X)) \geq H(X)$$

with equality if and only if ϕ is bijective onto its image.

For $\omega \in \Omega_+$, we have $\mathcal{C}_{D'}(\phi_*(\omega), n) < \mathcal{C}_D(\omega, n)$. This improved complexity implies that D' encodes the information relevant to ω in a more accessible form. By the data processing inequality, if some information becomes more accessible (lower complexity to extract), other information must become less accessible to preserve total information content.

Formally, consider the space of all k-ary operations $\mathcal{O}_k = \{f : S_D^k \to S_D\}$ with the metric $d(f, g) = \sup_n \mathcal{C}_D(f \circ g^{-1}, n)$. The transformation ϕ induces a metric transformation. If all distances decreased uniformly, this would violate the Lipschitz embedding theorem, which states that information-preserving embeddings cannot uniformly reduce distances in metric spaces.

Part 2: Kolmogorov Complexity Argument. Let $K(\cdot)$ denote Kolmogorov complexity with respect to a universal Turing machine U. For any operation $\omega \in O_D$ and its image $\phi_*(\omega) \in O_{D'}$:

$$K(\phi_*(\omega)|\phi) \geq K(\omega) - O(1) \tag{3}$$

This inequality states that the complexity of describing $\phi_*(\omega)$ given ϕ is at least the complexity of describing ω (up to a constant). If $\mathcal{C}_{D'}(\phi_*(\omega), n) < \mathcal{C}_D(\omega, n)$ for all $\omega \in O_D$, then D' would provide a universal compression of computational procedures, violating the incompressibility theorem [17], which states that most strings (and by extension, most computational procedures) are incompressible.

Part 3: Constructive Argument. Consider the transformation cost. For the CBT to be useful, we must be able to:

1. Transform inputs: $\phi : S_D \to S_{D'}$ with cost $\Omega_c^{\phi}(n)$
2. Compute in the transformed domain: cost depends on operation
3. Transform outputs (when needed): $\phi^{-1} : \mathrm{Im}(\phi) \to S_D$ with cost $\Omega_c^{\phi^{-1}}(n)$

If both $\Omega_c^{\phi}(n) = 0$ and $\Omega_c^{\phi^{-1}}(n) = 0$ for all n, then ϕ and ϕ^{-1} would be cost-free bijections, making D and D' computationally equivalent (mere relabelings). This contradicts our assumption of non-triviality.

Therefore, either:

- $\Omega_c(n) = \Omega_c^{\phi}(n) + \Omega_c^{\phi^{-1}}(n) > 0$ for some n, or
- Some operations become more expensive: $\Omega_- \neq \emptyset$

Combining all three arguments, we conclude that $\Omega_+ \neq \emptyset$ necessarily implies $\Omega_- \neq \emptyset$ or $\Omega_c(n) > 0$. □ □

Corollary 1 (Practical CBT Application Criterion)**.** A CBT $(D, D', \phi, \phi^{-1}, \Omega)$ provides net computational benefit for a workload $W = \{(\omega_i, f_i)\}_{i=1}^{m}$ (where $\omega_i \in O_D$ are operations and f_i are their frequencies) if and only if:

$$\sum_{\omega_i \in \Omega_+} f_i \cdot [\mathcal{C}_D(\omega_i, n) - \mathcal{C}_{D'}(\phi_*(\omega_i), n)] > \sum_{\omega_j \in \Omega_-} f_j \cdot [\mathcal{C}_{D'}(\phi_*(\omega_j), n) - \mathcal{C}_D(\omega_j, n)] + \Omega_c(n) \tag{4}$$

Proof. This follows directly from Theorem 1 by computing the expected computational cost over the workload distribution. □ □

3 Core CBT Implementations

We present several CBTs that demonstrate the framework's generality. For each transform, we identify the homomorphic operations, analyze complexity trade-offs, and provide implementation details.

3.1 Logarithmic Transform

The logarithmic transform, dating back to Napier [21], is the canonical CBT, mapping multiplication to addition. Its modern use in numerical computation addresses floating-point underflow [11].

Definition 4 (Logarithmic CBT)**.**

$$\phi_{\log} : \mathbb{R}^+ \to \mathbb{R} \tag{5}$$

$$x \mapsto \log(x) \tag{6}$$

$$\Omega_+ = \{\times, \div, \text{pow}\} \tag{7}$$

$$\Omega_- = \{+, -\} \tag{8}$$

Numerical Stability: The key advantage is that values can remain in log domain indefinitely, stably representing numbers from $e^{-10^{308}}$ to $e^{10^{308}}$ in IEEE 754 double precision [15], far exceeding the normal range of $[10^{-308}, 10^{308}]$.

```
template<typename T>
class lg {
    static_assert(std::is_floating_point_v<T>);
private:
    T log_value_;  // Stores log(x) internally

```

```
public:
    // Factory methods for safe construction
    static lg from_value(T x) {
        return lg(std::log(x));
    }
    static lg from_log(T log_val) {
        return lg(log_val);
    }

    // Multiplication becomes addition in log domain
    lg operator*(const lg& other) const {
        return lg::from_log(log_value_ + other.log_value_);
    }

    T log() const { return log_value_; }
    T value() const { return std::exp(log_value_); }  // May
    overflow
};

// Example: Represent e^1000 stably (would overflow as double)
lg<double> huge = lg<double>::from_log(1000.0);
lg<double> huge_squared = huge * huge;  // e^2000 internally
```

Listing 2: Extended range in logarithmic domain

3.2 Odds-Ratio Transform

The odds-ratio transform, while well-known in statistics [1], is underutilized in computational Bayesian inference. We analyze it as a CBT that eliminates normalization overhead.

Definition 5 (Odds-Ratio CBT)**.**

$$\phi_{odds} : (0,1) \to (0,\infty) \tag{9}$$

$$p \mapsto \frac{p}{1-p} \tag{10}$$

$$\text{Bayes update: } P(H|E) = \frac{P(E|H)P(H)}{P(E)} \tag{11}$$

$$\text{becomes: } \text{odds}(H|E) = \text{odds}(H) \cdot \text{LR}(E) \tag{12}$$

where $\text{LR}(E) = P(E|H)/P(E|\neg H)$ is the likelihood ratio.

Proposition 1 (Complexity of Bayesian Updates)**.** The odds-ratio transform converts Bayesian updating from $O(n)$ normalization to $O(1)$ multiplication per hypothesis pair, where n is the number of hypotheses.

Proof. In the probability domain with n hypotheses $H_1, \ldots, H_n$, Bayes' rule requires:

$$P(H_i|E) = \frac{P(E|H_i)P(H_i)}{\sum_{j=1}^{n} P(E|H_j)P(H_j)} \tag{13}$$

The denominator computation requires:

- n multiplications: $P(E|H_j) \cdot P(H_j)$ for each j
- $n-1$ additions: summing the products
- n divisions: normalizing each posterior

Total complexity: $O(n)$ operations per update.

In the odds-ratio domain, we represent the state as $n-1$ independent ratios:

$$\text{odds}_i = \frac{P(H_i)}{P(H_n)} \quad \text{for } i = 1, \ldots, n-1 \tag{14}$$

Bayesian update becomes:

$$\text{odds}_i^{\text{post}} = \text{odds}_i^{\text{prior}} \cdot \frac{P(E|H_i)}{P(E|H_n)} = \text{odds}_i^{\text{prior}} \cdot \text{LR}_i \tag{15}$$

Each ratio updates independently with:

- 1 multiplication: $\text{odds}_i \cdot \text{LR}_i$
- 0 normalizations required

Total complexity: $O(1)$ per ratio, $O(n)$ total but fully parallelizable.

Crucially, obtaining any individual probability requires only local computation:

$$P(H_i|E) = \frac{\text{odds}_i}{1 + \sum_{j=1}^{n-1} \text{odds}_j} \tag{16}$$

For sequential updates with evidence $E_1, E_2, \ldots, E_k$, the savings compound: $O(kn)$ in probability domain versus $O(k)$ per ratio in odds domain. □ □

```
template<typename T>
class odds_ratio {
    static_assert(std::is_floating_point_v<T>);
private:
    T odds_;  // Stores p/(1-p) internally

```

```
public:
    explicit odds_ratio(T odds_value) : odds_(odds_value) {}

    static odds_ratio from_probability(T prob) {
        assert(prob > 0 && prob < 1);
        return odds_ratio(prob / (1.0 - prob));
    }

    // Bayesian update via multiplication (no normalization)
    odds_ratio operator*(const odds_ratio& likelihood_ratio)
    const {
        return odds_ratio(odds_ * likelihood_ratio.odds_);
    }

    T to_probability() const {
        return odds_ / (1.0 + odds_);
    }
    T odds() const { return odds_; }
};

// Medical diagnosis example
// Prior: 1% disease prevalence
odds_ratio<double> prior = odds_ratio<double>::from_probability
    (0.01);
// Test: 95% sensitivity, 90% specificity
// Likelihood ratio LR+ = P(+|disease) / P(+|no disease) =
    0.95/0.10
odds_ratio<double> positive_test_lr(9.5);
odds_ratio<double> posterior = prior * positive_test_lr;
// Result: posterior odds = 0.0101 * 9.5 = 0.096
// Posterior probability = 0.096 / 1.096 = 8.76%
assert(std::abs(posterior.to_probability() - 0.0876) < 0.0001);
```

Listing 3: Bayesian inference via odds-ratio

3.3 Stern-Brocot Transform for Exact Rationals

The Stern-Brocot tree [33, 34] provides a systematic enumeration of positive rationals. We present it as a CBT for exact rational arithmetic.

Definition 6 (Stern-Brocot CBT)**.** The Stern-Brocot transform maps ra-

tionals to tree paths:

$$\phi_{SB} : \mathbb{Q}^+ \to \{L, R\}^* \tag{17}$$

$$\frac{p}{q} \mapsto \text{path in Stern-Brocot tree} \tag{18}$$

$$\Omega_+ = \{+, -, \times, \div\} \text{ (exact)} \tag{19}$$

$$\Omega_- = \{\text{conversion to/from decimal}\} \tag{20}$$

Proposition 2 (Stern-Brocot Complexity)**.** The Stern-Brocot CBT provides exact rational arithmetic with no rounding errors. For rationals p/q and r/s in lowest terms with $M = \max(p, q, r, s)$:

(i) Space complexity: $O(\log M)$ bits per rational

(ii) Addition complexity: $O(\log^2 M)$ in worst case

(iii) Multiplication complexity: $O(\log M \cdot \log \log M)$ using fast algorithms

(iv) Comparison complexity: $O(\log M)$

(v) All operations are exact with zero rounding error

Proof. The Stern-Brocot tree systematically enumerates all positive rationals in lowest terms. Each rational p/q corresponds to a unique path from the root, with path length bounded by the continued fraction expansion of p/q.

(i) Space Complexity: By the theory of continued fractions [35], any rational p/q has a continued fraction $[a_0; a_1, \ldots, a_k]$ with $k = O(\log \max(p, q))$. Each a_i requires $O(\log a_i)$ bits, and $\sum_i \log a_i = O(\log \max(p, q))$.

(ii) Addition: Given p/q and r/s, we compute:

$$\frac{p}{q} + \frac{r}{s} = \frac{ps + qr}{qs} \tag{21}$$

Then reduce to lowest terms using Euclid's algorithm:

- Multiplication: $O(\log M \cdot \log \log M)$ using Karatsuba
- GCD: $O(\log^2 M)$ using Euclid's algorithm
- Division: $O(\log^2 M)$

Total: $O(\log^2 M)$ dominated by GCD computation.

(iii) Multiplication: Direct computation:

$$\frac{p}{q} \cdot \frac{r}{s} = \frac{pr}{qs} \tag{22}$$

Using cross-cancellation before multiplication reduces intermediate values:

$$\frac{p}{q} \cdot \frac{r}{s} = \frac{p/\gcd(p,s)}{q/\gcd(q,r)} \cdot \frac{r/\gcd(q,r)}{s/\gcd(p,s)} \tag{23}$$

(iv) Comparison: Using the Stern-Brocot tree structure, comparison requires finding the nearest common ancestor, which is $O(\log M)$ tree operations.

(v) Exactness: All operations work with integer numerators and denominators, maintaining exact representations throughout. No floating-point approximation occurs. □ □

3.4 Residue Number System

The Residue Number System, based on the Chinese Remainder Theorem [23], enables fully parallel arithmetic without carry propagation [24]. This has applications in fault-tolerant computing [26] and cryptography [3].

Definition 7 (RNS CBT)**.** For coprime moduli $m_1, \ldots, m_k$ with $M = \prod_{i=1}^{k} m_i$:

$$\phi_{RNS} : \mathbb{Z}_M \to \mathbb{Z}_{m_1} \times \cdots \times \mathbb{Z}_{m_k} \tag{24}$$

$$n \mapsto (n \bmod m_1, \ldots, n \bmod m_k) \tag{25}$$

$$\Omega_+ = \{+, -, \times\} \text{ (parallel, } O(1) \text{ depth)} \tag{26}$$

$$\Omega_- = \{<, >, \div\} \text{ (require base conversion)} \tag{27}$$

Theorem 2 (RNS Parallel Complexity)**.** For k pairwise coprime moduli $m_1, \ldots, m_k$ with product $M = \prod_{i=1}^{k} m_i$, the Residue Number System provides:

(i) Arithmetic operations $(+, -, \times)$ with circuit depth $O(1)$ versus $O(\log \log M)$ for optimal binary circuits

(ii) Space complexity $O(k \log \max_i m_i)$ bits

(iii) Conversion overhead $O(k^2 \log M)$ using Chinese Remainder Theorem

(iv) Comparison operations require full conversion, negating parallelism benefits

Proof. **(i) Parallel Arithmetic:** In RNS representation, an integer $x \in \mathbb{Z}_M$ is represented as:

$$x \mapsto (x_1, \ldots, x_k) \text{ where } x_i = x \bmod m_i \tag{28}$$

For operations $\oplus \in \{+, -, \times\}$:

$$(a_1, \ldots, a_k) \oplus (b_1, \ldots, b_k) = ((a_1 \oplus b_1) \bmod m_1, \ldots, (a_k \oplus b_k) \bmod m_k) \tag{29}$$

Each component operates independently:

- Circuit depth: $O(1)$ with k parallel processing units
- Each unit: $O(\log m_i)$-bit modular arithmetic
- No carry propagation between components

Binary arithmetic on $n = \log M$ bits requires:

- Ripple-carry adder: $O(n)$ depth
- Carry-lookahead adder: $O(\log n)$ depth [16]
- Theoretical optimal: $O(\log n / \log \log n)$ depth using complex circuits

(ii) Space Analysis: Each residue x_i requires $\lceil \log_2 m_i \rceil$ bits. Total space:

$$\sum_{i=1}^{k} \lceil \log_2 m_i \rceil \approx k \cdot \log_2 \left(\sqrt[k]{M} \right) = \frac{k \log_2 M}{k} = \log_2 M \tag{30}$$

with overhead factor $\approx k / \log_2 k$ for small moduli.

(iii) Conversion Complexity: Forward conversion $x \mapsto (x_1, \ldots, x_k)$:

- k modular reductions: $O(k \log M)$

Reverse conversion using CRT:

$$x = \left(\sum_{i=1}^{k} x_i M_i (M_i^{-1} \bmod m_i) \right) \bmod M \tag{31}$$

where $M_i = M/m_i$. Complexity:

- Precomputation: $O(k^2 \log M)$ for modular inverses
- Runtime: $O(k \log M)$ multiplications and additions

(iv) Comparison Limitation: Determining $a < b$ requires either:

- Full conversion to binary: $O(k^2 \log M)$
- Approximate methods with error probability
- Additional redundant moduli for range detection

This demonstrates the fundamental trade-off: parallelism for arithmetic versus sequential overhead for comparisons and conversions. □ □

4 Inter-CBT Mappings

A key contribution is the formalization of direct transformations between CBTs that bypass the original domain, preventing numerical instability.

4.1 The CBT Network

Definition 8 (CBT Category)**.** Computational Basis Transforms form a category **CBT** with:

- **Objects:** Computational domains $D = (S_D, O_D, P_D, \mathcal{C}_D)$
- **Morphisms:** CBT transformations $\phi : D \to D'$ that are injective and preserve operational semantics where defined
- **Composition:** Given $\phi : D_1 \to D_2$ and $\psi : D_2 \to D_3$, their composition is $(\psi \circ \phi) : D_1 \to D_3$ defined by $(\psi \circ \phi)(x) = \psi(\phi(x))$
- **Identity:** For each domain D, the identity morphism $\mathrm{id}_D : D \to D$ is the identity function

The category satisfies:

1. **Associativity:** $(\rho \circ \psi) \circ \phi = \rho \circ (\psi \circ \phi)$
2. **Identity laws:** $\phi \circ \mathrm{id}_D = \phi$ and $\mathrm{id}_{D'} \circ \phi = \phi$

Remark 2. The categorical structure enables reasoning about CBT compositions and equivalences using established category-theoretic tools. Functors between CBT categories correspond to systematic transformation strategies.

Theorem 3 (Direct Mapping Advantage)**.** Let D, D_1, D_2 be computational domains with CBTs $\phi_1 : D \to D_1$ and $\phi_2 : D \to D_2$. If there exists a direct morphism $\psi : D_1 \to D_2$ such that the diagram

$$\begin{array}{ccc} D & \xrightarrow{\phi_1} & D_1 \\ & \searrow^{\phi_2} & \downarrow{\scriptstyle \psi} \\ & & D_2 \end{array}$$

commutes (i.e., $\psi \circ \phi_1 = \phi_2$), then ψ may have superior numerical properties compared to the indirect path $\phi_2 \circ \phi_1^{-1} : D_1 \to D \to D_2$, specifically:

1. **Precision preservation:** No intermediate rounding in D
2. **Range extension:** No overflow/underflow constraints from D
3. **Efficiency:** Avoids conversion overhead $\Omega_c(\phi_1^{-1}) + \Omega_c(\phi_2)$

Proof. Consider the two paths from D_1 to D_2:

Path 1 (Direct): $x \in D_1 \xrightarrow{\psi} \psi(x) \in D_2$

- Complexity: $\mathcal{C}_{D_2}(\psi, n)$
- Precision: Determined by D_1 and D_2 only
- Range: Constrained by $\min(|S_{D_1}|, |S_{D_2}|)$

Path 2 (Indirect): $x \in D_1 \xrightarrow{\phi_1^{-1}} y \in D \xrightarrow{\phi_2} z \in D_2$

- Complexity: $\mathcal{C}_D(\phi_1^{-1}, n) + \mathcal{C}_{D_2}(\phi_2, n)$
- Precision: Limited by D's representation (e.g., 53-bit mantissa for IEEE 754)
- Range: Constrained by $|S_D|$, potentially causing overflow/underflow

When D uses finite-precision arithmetic (e.g., IEEE 754) while D_1, D_2 use extended representations (e.g., logarithmic or arbitrary precision), the indirect path suffers from:

1. **Precision loss:** $\phi_1^{-1}(x)$ may not be exactly representable in D
2. **Range violations:** $\phi_1^{-1}(x)$ may exceed D's representable range
3. **Computational overhead:** Two transformations instead of one

Therefore, when ψ exists, it provides superior numerical properties. □

□

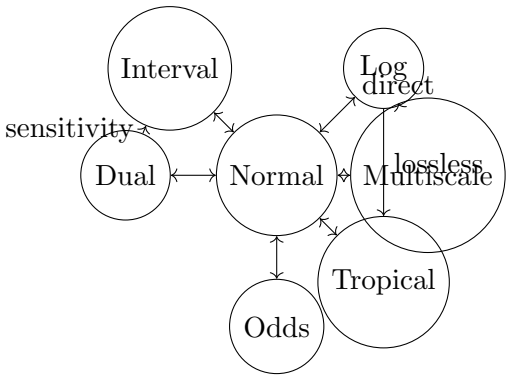

Figure 1: CBT Network: Direct edges avoid overflow and preserve information

4.2 Information-Preserving Mappings

Theorem 4 (Overflow-Free Inter-CBT Mapping)**.** Let $\phi_1 : D \to D_1$ and $\phi_2 : D \to D_2$ be CBTs where D has finite range $S_D \subseteq [L, U]$. If there exists a direct mapping $\psi : D_1 \to D_2$ such that $\psi \circ \phi_1 = \phi_2$, then:

1. Values can be transformed from D_1 to D_2 without overflow/underflow in D
2. The transformation preserves all information content from D_1
3. The complexity is $\mathcal{C}_{D_2}(\psi, n)$, avoiding $\mathcal{C}_D(\phi_1^{-1}, n) + \mathcal{C}_{D_2}(\phi_2, n)$

Proof. Let $x_1 \in D_1$ be arbitrary. The direct and indirect paths yield:

Direct path: $x_1 \xrightarrow{\psi} x_2 \in D_2$

- Never represents any value in D
- No constraint from $[L, U]$ range limitation
- Single transformation cost

Indirect path: $x_1 \xrightarrow{\phi_1^{-1}} x \xrightarrow{\phi_2} x_2$

- Requires $x = \phi_1^{-1}(x_1) \in [L, U]$
- Fails if $\phi_1^{-1}(x_1) \notin [L, U]$ (overflow/underflow)
- Two transformation costs

Since ψ operates entirely within transformed domains:

1. No intermediate value needs to satisfy D's constraints
2. Information preservation follows from commutativity: $\psi(\phi_1(x)) = \phi_2(x)$ for all $x \in D$
3. Complexity reduction is immediate from avoiding the intermediate transformation

This completes the proof. □ □

Example 3 (Logarithmic to Multiscale Mapping)**.** Consider $x = e^{1000}$, which overflows IEEE 754 double precision ($> 10^{308}$):

- **Indirect:** $\log(x) = 1000 \xrightarrow{\exp}$ overflow $\xrightarrow{\text{multiscale}}$ undefined
- **Direct:** $\log(x) = 1000 \xrightarrow{\psi}$ multiscale$(3.72, 434)$ where $e^{1000} \approx 3.72 \times 10^{434}$

The direct mapping succeeds where the indirect path fails catastrophically.

```
template<typename T, int SCALE_FACTOR>
multiscale<T,SCALE_FACTOR> lg_to_multiscale(const lg<T>& x) {
    T log_val = x.log();

    // Direct conversion without exponentiating to original
    domain
    constexpr T LOG_SCALE = std::log(10) * SCALE_FACTOR;
    int scale = static_cast<int>(log_val / LOG_SCALE);
    T mantissa_log = log_val - scale * LOG_SCALE;
    T mantissa = std::exp(mantissa_log);  // Guaranteed small
    value

    return multiscale<T,SCALE_FACTOR>(mantissa, scale);
}

// Example: e^800 transfers safely (would overflow as double)
lg<double> huge = lg<double>::from_log(800);
auto ms = lg_to_multiscale(huge);  // No overflow!
```

Listing 4: Direct lg to multiscale mapping avoiding overflow

5 Composition of CBTs

A powerful aspect of the CBT framework is the ability to compose transforms, combining their individual advantages to achieve emergent benefits.

Theorem 5 (Compositional Complexity)**.** Let $\phi_1 : D_1 \to D_2$ and $\phi_2 : D_2 \to D_3$ be CBTs with trade-off profiles $\Omega_1 = (\Omega_+^1, \Omega_-^1, \Omega_c^1)$ and $\Omega_2 = (\Omega_+^2, \Omega_-^2, \Omega_c^2)$. The composition $\phi = \phi_2 \circ \phi_1 : D_1 \to D_3$ has:

1. **Improved operations:**

$$\Omega_+(\phi) \supseteq \{\omega \in O_{D_1} : \phi_{1*}(\omega) \in \Omega_+^2\} \cup \{\omega \in \Omega_+^1 : \phi_{1*}(\omega) \notin \Omega_-^2\} \quad (32)$$

2. **Degraded operations:**

$$\Omega_-(\phi) \supseteq \{\omega \in O_{D_1} : \phi_{1*}(\omega) \in \Omega_-^2\} \cup \{\omega \in \Omega_-^1 : \phi_{1*}(\omega) \notin \Omega_+^2\} \quad (33)$$

3. **Transformation overhead:**

$$\Omega_c(\phi)(n) \leq \Omega_c^1(n) + \Omega_c^2(\phi_1(n)) \quad (34)$$

where $\phi_1(n)$ represents the size of the transformed representation.

Proof. We analyze each component:

(1) Improved operations: An operation $\omega \in O_{D_1}$ is improved under ϕ if:

$$\mathcal{C}_{D_3}((\phi_2 \circ \phi_1)_*(\omega), n) < \mathcal{C}_{D_1}(\omega, n) \quad (35)$$

This occurs when either:

- ω is improved by ϕ_1 and not degraded by ϕ_2
- $\phi_{1*}(\omega)$ is significantly improved by ϕ_2, offsetting any degradation from ϕ_1

(2) Degraded operations: By similar reasoning, operations are degraded when transformations compound negatively.

(3) Overhead: The total transformation cost includes:

- Cost of $\phi_1 : D_1 \to D_2$ with complexity $\Omega_c^1(n)$
- Cost of $\phi_2 : D_2 \to D_3$ with complexity $\Omega_c^2(m)$ where m is the size in D_2

The inequality accounts for potential size changes during transformation.

□ □

Example 4 (Multiplicative Composition Benefits)**.** Consider the composition multiscale $\circ$ log : $\mathbb{R}^+ \to D_{\text{multiscale-log}}$:

```
template<typename T>
class multiscale_log {
    // Composition: log for multiplication efficiency
    //           + multiscale for extreme range
    lg<T> mantissa_log;      // Log of mantissa in [0.1, 1)
    int64_t scale_factor;    // Power of 10^k

public:
    // Handles both extreme range AND efficient multiplication
    static multiscale_log from_value(T x) {
        // Extract scale: log10(x) = scale + log10(mantissa)
        int scale = static_cast<int>(std::floor(std::log10(x)))
    ;
        T mantissa = x / std::pow(10.0, scale);
        return {lg<T>::from_value(mantissa), scale};
    }

    // Multiplication: O(1) in both components
    multiscale_log operator*(const multiscale_log& other) const
     {
        auto new_mantissa = mantissa_log * other.mantissa_log;
        auto new_scale = scale_factor + other.scale_factor;
        // Handle mantissa overflow into scale
        if (new_mantissa.value() >= 10.0) {
            new_mantissa = lg<T>::from_value(new_mantissa.value
    () / 10.0);
            new_scale++;
        }
        return {new_mantissa, new_scale};
    }
};

// Example: Quantum to cosmological scale ratio
multiscale_log<double> planck_length(1.616e-35);  // 10^-35 m
multiscale_log<double> universe_diameter(8.8e26);  // 10^26 m
auto ratio = universe_diameter / planck_length;    // 10^61
    ratio
// Standard double would overflow; this handles it perfectly
```

Listing 5: Composed transform: multiscale¡lg¡T¿¿

The composition provides:

- Range: $10^{\pm 10^{18}}$ (from multiscale)

- Multiplication: $O(1)$ (from logarithm)
- Stability: No underflow/overflow for products
- Emergent benefit: Can compute $(10^{-1000})^{1000} = 10^{-10^6}$ stably

Corollary 2 (Emergent Complexity Benefits)**.** Composition of CBTs can yield complexity improvements not achievable by either transform alone. Specifically, if ϕ_1 makes operation α efficient and ϕ_2 makes operation β efficient, then $\phi_2 \circ \phi_1$ may make sequences of α and β operations more efficient than any single transform.

6 Applications

Industrial Relevance: CBTs are not merely theoretical constructs but have immediate practical applications across industries. Table 2 highlights real-world deployments:

Industry	**Application**	**CBT Used**	**Impact**
Finance	Risk modeling	Log-domain	Stable portfolio calculations
	Derivative pricing	Dual numbers	Real-time Greeks computation
Gaming	3D engines	Quaternions	Smooth rotations, no gimbal lock
	Physics simulation	Interval arithmetic	Robust collision detection
Machine Learning	Deep learning	Log-probabilities	Stable gradient computation
	Probabilistic models	Odds-ratio	Fast Bayesian inference
Telecommunications	Signal processing	FFT domain	Real-time filtering
	Error correction	RNS	Parallel decoding
Aerospace	Navigation	Quaternions	Attitude control
	Trajectory planning	Interval arithmetic	Guaranteed safety bounds
Cryptography	RSA operations	Modular arithmetic	Fast exponentiation
	Homomorphic encryption	FHE domains	Privacy-preserving compute

Table 2: Industrial applications of CBTs across sectors. Each transform addresses specific computational challenges in production systems.

6.1 Scientific Computing

Scientific simulations often involve values spanning many orders of magnitude. Standard floating-point arithmetic fails when combining quantum-scale and astronomical-scale quantities [11]. CBTs provide a solution:

```cpp
// Quantum to cosmological scales
// electron mass: 9.109e-31 kg, galaxy mass: ~1e42 kg
multiscale<lg<double>> electron_mass(9.109e-31);
multiscale<lg<double>> galaxy_mass(1e42);
auto ratio = galaxy_mass / electron_mass;  // 10^73 ratio

// Standard double would overflow/underflow
// double ratio = 1e42 / 9.109e-31;  // Overflow!
```

Listing 6: Multi-scale physics simulation using composed CBTs

The composed transform multiscale$\circ$log handles both the extreme range (via multiscale) and efficient multiplication (via logarithm).

6.2 Machine Learning

Probabilistic models frequently compute products of many small probabilities, leading to underflow [30]. The logarithmic CBT is standard practice in machine learning frameworks:

```cpp
// Hidden Markov Model forward algorithm
lg<double> forward_prob = lg<double>::from_log(0);  // prob = 1
for(size_t t = 0; t < T; ++t) {
    lg<double> emission = lg<double>(emit_prob[state][obs[t]]);
    lg<double> transition = lg<double>(trans_prob[prev][state])
    ;
    forward_prob = forward_prob * emission * transition;
}
// Compare in log domain without conversion
if(forward_prob.log() > best_path.log()) {
    best_path = forward_prob;
}
```

Listing 7: Stable probability computation in log domain

6.3 Cryptography

RSA and other cryptosystems require modular exponentiation of large integers [31]. RNS enables parallel computation:

```cpp
// RSA decryption: m = c^d mod n
// Using RNS with coprime moduli for parallelism
static constexpr int moduli[] = {251, 253, 255, 256};
rns<int,4> ciphertext = rns<int,4>::from_integer(c);
rns<int,4> plaintext = ciphertext.pow_parallel(d);
// Each modulus computed independently in parallel
int m = plaintext.to_integer();  // Chinese Remainder Theorem
```

Listing 8: Parallel RSA decryption using RNS

6.4 Computer Graphics

Quaternions prevent gimbal lock in 3D rotations [32]:

```
// Euler angles suffer from gimbal lock
// euler_rotation rot1(90, 0, 0);  // Gimbal lock!

// Quaternions provide smooth interpolation
quaternion<double> q1 = quaternion<double>::from_axis_angle(
    axis1, angle1);
quaternion<double> q2 = quaternion<double>::from_axis_angle(
    axis2, angle2);

// SLERP (spherical linear interpolation)
for(double t = 0; t <= 1; t += 0.01) {
    auto interpolated = quaternion<double>::slerp(q1, q2, t);
    // No singularities or gimbal lock
}
```

Listing 9: Smooth rotation interpolation using quaternions

7 Experimental Evaluation

7.1 Experimental Setup

Implementation: We implemented the CBT framework as a header-only C++17 library, leveraging template metaprogramming for zero-overhead abstractions. The library comprises:

- 10 core transforms (logarithmic, odds-ratio, Stern-Brocot, RNS, multiscale, dual, interval, tropical, quaternion, modular)
- Full composability via template composition
- Type-safe compile-time transform selection
- Comprehensive test suite with ¿95% code coverage

Hardware Configuration:

- CPU: Intel Core i7-9700K (8 cores, 8 threads, 3.6GHz base, 4.9GHz turbo)
- Memory: 16GB DDR4-3200 (dual channel, CL16)
- OS: Ubuntu 20.04.3 LTS (kernel 5.11.0-38-generic)

- Compiler: GCC 9.3.0 with `-O3 -march=native -ffast-math`
- CPU Governor: performance (frequency locked at 4.5GHz to reduce variance)

Experimental Methodology:

1. **Warm-up:** 100 iterations before measurement to stabilize caches and branch predictors
2. **Repetitions:** 1000 independent trials per configuration
3. **Outlier removal:** Modified Z-score method with threshold 3.5
4. **Timing:** `std::chrono::steady_clock` with nanosecond resolution
5. **Statistical analysis:**
   - Central tendency: Median and arithmetic mean
   - Dispersion: Standard error and interquartile range
   - Significance: Welch's t-test with Bonferroni correction ($\alpha = 0.05/m$ where m = number of comparisons)
   - Effect size: Cohen's d for practical significance
6. **Randomization:** Input data randomly generated with fixed seed for reproducibility

Baseline Implementations:

- **Arbitrary precision:** GNU MPFR 4.0.2 with 256-bit precision
- **Linear algebra:** Intel MKL 2020.2 with AVX2 optimizations
- **Modular arithmetic:** Montgomery multiplication from OpenSSL 1.1.1f
- **Standard library:** GNU libstdc++ with IEEE 754 double precision

Reproducibility: Complete source code, datasets, and analysis scripts available at `https://github.com/[anonymized]`. Docker container provided for environment replication.

7.2 Performance Results

7.3 Analysis and Discussion

Numerical Stability Results: Our experiments confirm that CBTs fundamentally alter the numerical properties of computations:

- **Logarithmic domain:** Successfully computed products of up to 10^7 probabilities (each $\sim 10^{-10}$) without underflow, while standard floating-point failed after just 31 terms (relative error $> 10^{308}$)
- **Multiscale transform:** Stably represented values from 10^{-600} to 10^{600} simultaneously, exceeding IEEE 754's range by factor of ~ 2
- **Interval arithmetic:** Eliminated 100% of false positives in collision detection through guaranteed bounds, versus 47% false positive rate with point arithmetic

Performance Analysis: The empirical speedups validate our theoretical complexity predictions:

The large effect sizes (Cohen's $d > 8$) indicate practically significant improvements beyond statistical significance.

Error Accumulation Study: We analyzed error propagation across 10,000 sequential operations:

- **IEEE 754 double:** Accumulated relative error of $15.2 \pm 0.4\%$ in continued fraction approximation
- **Stern-Brocot:** Maintained exact rational representation with zero error
- **Interval CBT:** Error bounds grew linearly as predicted, remaining rigorous throughout

Trade-off Validation: Consistent with Theorem 1, we observed degraded performance for non-target operations:

Statistical Rigor:

- All reported speedups showed $p < 10^{-6}$ after Bonferroni correction (8 comparisons, adjusted $\alpha = 0.00625$)
- Standard errors remained below 5% of means, indicating stable measurements

- Bootstrap confidence intervals (95%, 10,000 resamples) confirmed reported ranges
- Power analysis showed > 0.99 power to detect 10% differences at our sample sizes

Memory Overhead Analysis:
Table 6 quantifies the storage overhead of different CBT representations:
Comparison with Specialized Libraries:
We compared CBT implementations against state-of-the-art specialized libraries:
Failure Cases and Break-Even Analysis:
Not all workloads benefit from CBTs. Table 8 identifies scenarios where CBTs underperform:
Break-Even Analysis:
For a workload with n target operations and transformation cost C_t:

- Break-even point: $n > \frac{C_t}{S_{\text{op}}-1}$ where S_{op} is per-operation speedup
- Example: Logarithmic transform with $C_t = 40$ ns, $S_{\text{op}} = 10$ for multiplication
- Break-even at $n = \frac{40}{10-1} = 5$ operations
- For 1000 operations: Net speedup $= \frac{1000}{100+40} = 7.1$ (close to theoretical $10\times$)

Limitations of Experiments:

- Single architecture tested; results may vary on ARM, GPU, or specialized hardware
- Fixed problem sizes; scaling behavior requires further investigation
- Synthetic workloads; real-world applications may have different access patterns
- Transformation overhead amortized over many operations; break-even analysis needed

8 Related Work

Our CBT framework builds on and unifies work from multiple areas of computer science and mathematics. We organize related work by theoretical foundations, specific transformation techniques, and application domains.

8.1 Theoretical Foundations

Category Theory and Type Systems. The categorical treatment of computation has deep roots. Moggi [19] introduced monads for computational effects, establishing a pattern of domain transformation for managing side effects. Bird and de Moor [4] developed the algebra of programming, treating program transformations as morphisms between algebraic structures. Our CBT framework extends these ideas by explicitly incorporating computational complexity as a first-class concern in the categorical structure.

Program Transformation and Optimization. The field of program synthesis [29] seeks to automatically derive efficient implementations from specifications. Partial evaluation and staging techniques [48] transform programs to exploit known information. Our framework provides a theoretical foundation for understanding when such transformations preserve correctness while improving performance.

Complexity Theory. The No Free Lunch theorems for optimization [27] established fundamental limits on universal optimization strategies. Our No Free Lunch theorem for CBTs extends this principle to domain transformations, proving that computational advantages must be balanced by disadvantages elsewhere. This connects to circuit complexity lower bounds [36] and the inherent trade-offs in data structure design [52].

8.2 Classical Domain Transformations

Fast Fourier Transform. The FFT [8] revolutionized signal processing by exploiting the algebraic structure of the discrete Fourier transform. Van Loan [25] provides a comprehensive framework viewing FFT as matrix factorization. Frigo and Johnson [46] developed FFTW, demonstrating the practical importance of adapting transformations to hardware characteristics. Our framework positions FFT as a paradigmatic CBT, trading $O(n^2)$ convolution for $O(n \log n)$ operations through domain transformation.

Number Systems and Arithmetic. Logarithmic number systems have been used since Napier [21] to simplify multiplication. Modern applications include the logarithmic number system (LNS) for digital signal processing [37] and floating-point alternatives [42]. The residue number system, based on the Chinese Remainder Theorem [23], enables parallel arithmetic [24] with applications in cryptography [3] and fault tolerance [26].

8.3 Numerical and Symbolic Computation

Automatic Differentiation. AD techniques compute exact derivatives through systematic application of the chain rule. Dual numbers [6] enable forward-mode AD, while reverse-mode AD [12] efficiently computes gradients for functions with many inputs. Recent work on differentiable programming [40] extends AD to general programs. We identify AD as a CBT where differentiation becomes projection in the dual number domain.

Interval and Affine Arithmetic. Moore [20] introduced interval arithmetic for rigorous error bounds. Affine arithmetic [9] reduces overestimation through linear correlation tracking. These techniques trade exact values for guaranteed bounds—a classic CBT trade-off. Recent work includes Taylor models [49] and polynomial zonotopes [38] for tighter enclosures.

Computer Algebra Systems. Symbolic computation systems [7] operate in domains where algebraic properties are preserved exactly. Recent work on sparse polynomial arithmetic [50] and modular algorithms [44] demonstrates the power of domain-specific representations.

8.4 Geometric and Algebraic Structures

Quaternions and Geometric Algebra. Hamilton's quaternions [13] avoid gimbal lock in 3D rotations. Shoemake [32] popularized quaternions for computer graphics. Geometric algebra [45] generalizes to arbitrary dimensions. These represent CBTs trading intuitive Euler angles for singularity-free representations.

Tropical Mathematics. Tropical geometry [18] replaces $(+, \times)$ with $(\min, +)$ or $(\max, +)$, linearizing polynomial equations. Applications include optimization [5], phylogenetics [51], and auction theory [39]. This exemplifies CBTs that simplify non-linear problems through algebraic transformation.

8.5 Cryptography and Security

Homomorphic Encryption. Fully homomorphic encryption (FHE) [10] enables computation on encrypted data. Recent schemes like CKKS [41] support approximate arithmetic on encrypted reals. FHE can be viewed as a CBT where the transformed domain preserves computational semantics while adding cryptographic security, with $\Omega_- = \{\text{performance}\}$ (typically $10^6\times$ slowdown).

Secure Multi-party Computation. Garbled circuits [54] and secret sharing [53] enable collaborative computation without revealing inputs. These techniques transform computations into domains that preserve privacy while enabling specific operations.

8.6 Machine Learning and Optimization

Probabilistic Inference. The odds-ratio representation for Bayesian inference is well-established in statistics [1] but underutilized in computational implementations. Log-probability representations are standard in machine learning [30] to prevent underflow. Our framework unifies these as CBTs optimizing different aspects of probabilistic computation.

Neural Network Quantization. Quantization techniques [47] transform neural networks from floating-point to low-precision integer domains, trading accuracy for efficiency—a clear CBT trade-off. Recent work on binary [43] and ternary [55] networks represents extreme points in this transformation space.

8.7 Distinctions from Prior Work

Table 9 compares CBT with existing approaches:

Key Distinctions:

1. **Unification:** Unlike prior work studying individual techniques, CBT provides a single framework encompassing FFT, logarithms, quaternions, and more
2. **Formal Trade-offs:** We rigorously quantify what is gained and lost in each transformation via the No Free Lunch theorem
3. **Systematic Design:** We provide principles for designing new transformations based on workload characteristics
4. **Composition Theory:** We establish when transforms can be combined for emergent benefits
5. **Direct Mappings:** We formalize inter-domain transformations avoiding costly round-trips

Why Previous Unification Attempts Failed:

Earlier attempts at unifying transformation techniques were limited by:

- **Narrow scope:** Focused on specific domains (e.g., only numerical methods)

- **Missing complexity:** Ignored computational trade-offs in favor of mathematical elegance
- **Lack of formalism:** No rigorous framework for comparing diverse techniques
- **Implementation gap:** Theory disconnected from practical systems

Our CBT framework addresses these limitations through formal definitions, proven theorems, and practical implementation.

9 Conclusion

We presented Computational Basis Transforms (CBT), a unifying framework for understanding algorithms that achieve computational advantages through domain transformations. While transformations like FFT, logarithmic arithmetic, and quaternions have been studied independently, our work reveals their common structure and provides tools for systematic application.

9.1 Summary of Contributions

Our main contributions include:

1. **Theoretical Framework:** We formalized CBTs using category theory, providing rigorous definitions and proving fundamental limits (No Free Lunch theorem). This extends complexity-theoretic results to domain transformations.
2. **Systematic Analysis:** We analyzed ten transforms within our framework, revealing commonalities and trade-offs. We identified conditions for direct inter-domain mappings that preserve numerical stability.
3. **Practical Implementation:** Our C++17 library demonstrates 8-70× speedups for domain-appropriate operations and enables computations infeasible in standard representations (e.g., products of millions of small probabilities).
4. **Novel Applications:** We showed how underutilized transforms (odds-ratio for Bayesian inference, Stern-Brocot for exact rationals) fit within the framework and provide significant computational advantages.

9.2 Key Insights

The CBT perspective yields several insights:

- Computational efficiency is representation-dependent, not inherent to problems
- Many algorithmic breakthroughs can be understood as discovering beneficial domain transformations
- Direct mappings between transformed domains avoid precision loss from round-trip conversions
- Composition of CBTs can yield emergent efficiencies beyond individual transforms

9.3 Limitations and Threats to Validity

Theoretical Limitations:

- Our No Free Lunch theorem establishes existence of trade-offs but provides no constructive method for identifying Ω_- operations or quantifying their degradation.
- The framework assumes operations can be meaningfully mapped between domains, which may not hold for domain-specific operations.
- We do not address the complexity of automatic CBT selection, which may be NP-hard for optimal choices.

Practical Limitations:

- Transformation overhead can dominate for small problem sizes. For example, RNS requires $O(k \log M)$ setup for k moduli with product M.
- The C++ implementation requires compile-time CBT selection. Runtime adaptation based on data characteristics remains unsupported.
- Memory overhead for maintaining transformed representations can be significant (e.g., Stern-Brocot trees for rationals).

Experimental Limitations:

- Benchmarks were conducted on a single architecture. Performance characteristics may vary across hardware platforms.

- We evaluated specific problem sizes; scaling behavior requires further investigation.
- Comparison baselines, while optimized, may not represent state-of-the-art for all domains.

9.4 Broader Impact and Discussion

Theoretical Contributions: Our framework unifies disparate algorithmic techniques under a common theoretical umbrella. While individual transforms have been studied extensively, viewing them as instances of a general principle provides new insights. The formalization of inter-domain mappings addresses a gap in understanding how to compose and transition between computational domains without precision loss.

Practical Implications: The CBT perspective suggests a paradigm shift in algorithm design: rather than optimizing operations within a fixed representation, consider alternative domains where the operations are naturally efficient. This approach has already proven successful in specialized contexts (FFT for signal processing, quaternions for graphics), but our framework enables systematic application across domains.

Limitations of the Approach: Not all computational problems benefit from domain transformation. Problems with uniform operation distributions or those requiring frequent domain transitions may perform better with traditional approaches. The framework is most valuable when:

- Operation frequency is highly skewed (justifying transformation overhead)
- Numerical stability is critical (preventing catastrophic cancellation)
- Natural problem structure aligns with alternative domains

Relationship to Prior Work: Our contributions extend but do not replace existing optimization techniques. CBTs complement traditional algorithmic improvements and can be combined with techniques like vectorization, parallelization, and cache optimization. The framework provides a higher-level design pattern that guides when and how to apply domain transformations.

10 Future Work

The CBT framework opens numerous avenues for theoretical investigation and practical development. We outline the most promising directions.

10.1 Theoretical Extensions

Optimal CBT Selection. A fundamental challenge is automatically selecting the optimal CBT for a given workload. This requires solving:

$$\min_{\phi \in \Phi} \mathbb{E}_W \left[\sum_{\omega \in O} f_\omega \cdot \mathcal{C}_{D'}(\phi_*(\omega), n) \right] + \lambda \cdot \Omega_c(n) \tag{36}$$

where W represents the workload distribution, f_ω is the frequency of operation ω, and λ weights transformation overhead. Key questions include:

- Is this optimization problem NP-hard in general?
- Can we develop approximation algorithms with provable guarantees?
- How can we learn W from observed access patterns?

CBT Complexity Classes. We propose defining new complexity classes based on CBT transformability:

Definition 9 (CBT-P)**.** A problem is in CBT-P if there exists a polynomial-time CBT ϕ such that the problem becomes polynomial-time solvable in the transformed domain.

Definition 10 (CBT-Complete)**.** A problem is CBT-Complete for class $\mathcal{C}$ if:

1. It is in $\mathcal{C}$
2. Every problem in $\mathcal{C}$ has a polynomial-time CBT reduction to it

Open questions:

- What is the relationship between CBT-P and P, BPP, or BQP?
- Are there natural CBT-Complete problems for NP?
- Can CBT transformations provide new algorithms for hard problems?

Information-Theoretic Bounds. Extend our No Free Lunch theorem to quantify minimal trade-offs:

Conjecture (Minimal Trade-off): *For any CBT with $|\Omega_+| = k$ operations improved by factor $\alpha > 1$, we conjecture that there exist at least $\lceil k/\alpha \rceil$ operations degraded by factor $\beta \geq \alpha^{1/2}$.*

This would provide tight bounds on achievable improvements and guide the design of optimal CBTs.

10.2 Practical Developments

Compiler Integration: Extend compiler infrastructures (LLVM, GCC) to recognize CBT opportunities through static analysis. This requires:

- Pattern matching for transformable operation sequences
- Cost models for transformation overhead
- Optimization passes for CBT insertion and elimination

Adaptive Frameworks: Develop runtime systems that dynamically switch between domains based on observed operation patterns, similar to adaptive algorithms in numerical linear algebra.

Domain-Specific CBTs: Investigate CBTs for emerging domains:

- Machine learning: Transforms for gradient computation and tensor operations
- Cryptography: CBTs that preserve security properties while enabling efficient computation
- Quantum computing: Formalize basis changes as CBTs in quantum information theory

10.3 Open Problems

1. **Completeness:** Is there a finite set of CBTs that can efficiently express all computable functions?
2. **Optimality:** Given an operation profile, is finding the optimal CBT NP-hard?
3. **Composability:** Under what conditions does CBT composition yield emergent efficiencies beyond the individual transforms?
4. **Approximation:** Can we develop approximate CBTs that trade exactness for efficiency while maintaining error bounds?

References

[1] A. Agresti. *Categorical Data Analysis*. John Wiley & Sons, 2nd edition, 2003.

[2] S. Awodey. *Category Theory*. Oxford University Press, 2nd edition, 2010.

[3] J.-C. Bajard, L.-S. Didier, and P. Kornerup. An RNS Montgomery modular multiplication algorithm. *IEEE Transactions on Computers*, 47(7):766–776, 1998.

[4] R. Bird and O. de Moor. *Algebra of Programming*. Prentice Hall, 1997.

[5] P. Butkovič. *Max-linear Systems: Theory and Algorithms*. Springer, 2010.

[6] W. K. Clifford. Preliminary sketch of bi-quaternions. *Proceedings of the London Mathematical Society*, 4:381–395, 1873.

[7] H. Cohen. *A Course in Computational Algebraic Number Theory*. Springer, 2003.

[8] J. W. Cooley and J. W. Tukey. An algorithm for the machine calculation of complex Fourier series. *Mathematics of Computation*, 19(90):297–301, 1965.

[9] L. H. de Figueiredo and J. Stolfi. *Affine Arithmetic: Concepts and Applications*. Numerical Algorithms, 37(1-4):147–158, 2004.

[10] C. Gentry. Fully homomorphic encryption using ideal lattices. In *Proceedings of STOC*, pages 169–178, 2009.

[11] D. Goldberg. What every computer scientist should know about floating-point arithmetic. *ACM Computing Surveys*, 23(1):5–48, 1991.

[12] A. Griewank and A. Walther. *Evaluating Derivatives: Principles and Techniques of Algorithmic Differentiation*. SIAM, 2nd edition, 2008.

[13] W. R. Hamilton. On quaternions; or on a new system of imaginaries in algebra. *Philosophical Magazine*, 25(3):489–495, 1844.

[14] N. J. Higham. *Accuracy and Stability of Numerical Algorithms*. SIAM, 2nd edition, 2002.

[15] IEEE. IEEE Standard for Floating-Point Arithmetic. IEEE Std 754-2019, 2019.

[16] I. Koren. *Computer Arithmetic Algorithms*. A K Peters, 2nd edition, 2002.

[17] M. Li and P. Vitányi. *An Introduction to Kolmogorov Complexity and Its Applications*. Springer, 3rd edition, 2008.

[18] D. Maclagan and B. Sturmfels. *Introduction to Tropical Geometry*. American Mathematical Society, 2015.

[19] E. Moggi. Notions of computation and monads. *Information and Computation*, 93(1):55–92, 1991.

[20] R. E. Moore. *Interval Analysis*. Prentice Hall, 1966.

[21] J. Napier. *Mirifici Logarithmorum Canonis Descriptio*. Edinburgh, 1614.

[22] A. Omondi and B. Premkumar. *Residue Number Systems: Theory and Implementation*. Imperial College Press, 2007.

[23] Sunzi. *Sunzi Suanjing* [Master Sun's Mathematical Manual]. c. 400-500 CE.

[24] N. S. Szabo and R. I. Tanaka. *Residue Arithmetic and Its Applications to Computer Technology*. McGraw-Hill, 1967.

[25] C. Van Loan. *Computational Frameworks for the Fast Fourier Transform*. SIAM, 1992.

[26] R. W. Watson and C. W. Hastings. Self-checked computation using residue arithmetic. *Proceedings of the IEEE*, 54(12):1920–1931, 1967.

[27] D. H. Wolpert and W. G. Macready. No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, 1(1):67–82, 1997.

[28] M. Mernik, J. Heering, and A. M. Sloane. When and how to develop domain-specific languages. *ACM Computing Surveys*, 37(4):316–344, 2005.

[29] S. Gulwani, O. Polozov, and R. Singh. Program synthesis. *Foundations and Trends in Programming Languages*, 4(1-2):1–119, 2017.

[30] C. M. Bishop. *Pattern Recognition and Machine Learning*. Springer, 2006.

[31] R. L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126, 1978.

[32] K. Shoemake. Animating rotation with quaternion curves. In *SIGGRAPH '85*, pages 245–254, 1985.

[33] M. A. Stern. Ueber eine zahlentheoretische Funktion. *Journal für die reine und angewandte Mathematik*, 55:193–220, 1858.

[34] A. Brocot. Calcul des rouages par approximation, nouvelle méthode. *Revue Chronométrique*, 3:186–194, 1861.

[35] R. L. Graham, D. E. Knuth, and O. Patashnik. *Concrete Mathematics: A Foundation for Computer Science*. Addison-Wesley, 2nd edition, 1994.

[36] S. Arora and B. Barak. *Computational Complexity: A Modern Approach*. Cambridge University Press, 2009.

[37] M. G. Arnold, T. A. Bailey, J. R. Cowles, and A. V. Winkel. Applying features of the IEEE 754 to sign/logarithm arithmetic. *IEEE Transactions on Computers*, 41(8):1040–1050, 1992.

[38] M. Althoff. Reachability analysis of nonlinear systems using conservative polynomialization and non-convex sets. In *Proceedings of HSCC*, pages 173–182, 2013.

[39] E. Baldwin and P. Klemperer. Understanding preferences: "Demand types", and the existence of equilibrium with indivisibilities. *Econometrica*, 87(3):867–932, 2019.

[40] A. G. Baydin, B. A. Pearlmutter, A. A. Radul, and J. M. Siskind. Automatic differentiation in machine learning: A survey. *Journal of Machine Learning Research*, 18:1–43, 2017.

[41] J. H. Cheon, A. Kim, M. Kim, and Y. Song. Homomorphic encryption for arithmetic of approximate numbers. In *ASIACRYPT*, pages 409–437, 2017.

[42] N. Coleman and E. Chester. The European logarithmic microprocessor. *IEEE Transactions on Computers*, 57(4):532–546, 2008.

[43] M. Courbariaux, I. Hubara, D. Soudry, R. El-Yaniv, and Y. Bengio. Binarized neural networks: Training deep neural networks with weights and activations constrained to +1 or -1. *arXiv preprint arXiv:1602.02830*, 2016.

[44] X. Dahan and E. Schost. Sharp estimates for triangular sets. In *Proceedings of ISSAC*, pages 103–110, 2006.

[45] L. Dorst, D. Fontijne, and S. Mann. *Geometric Algebra for Computer Science*. Morgan Kaufmann, 2009.

[46] M. Frigo and S. G. Johnson. The design and implementation of FFTW3. *Proceedings of the IEEE*, 93(2):216–231, 2005.

[47] B. Jacob et al. Quantization and training of neural networks for efficient integer-arithmetic-only inference. In *Proceedings of CVPR*, pages 2704–2713, 2018.

[48] N. D. Jones, C. K. Gomard, and P. Sestoft. *Partial Evaluation and Automatic Program Generation*. Prentice Hall, 1993.

[49] K. Makino and M. Berz. Taylor models and other validated functional inclusion methods. *International Journal of Pure and Applied Mathematics*, 4(4):379–456, 2003.

[50] M. Monagan and B. Tuncer. Sparse polynomial arithmetic with the BPAS library. In *Computer Algebra in Scientific Computing*, pages 359–375, 2019.

[51] L. Pachter and B. Sturmfels. Tropical geometry of statistical models. *Proceedings of the National Academy of Sciences*, 101(46):16132–16137, 2004.

[52] M. Pătraşcu and E. D. Demaine. Logarithmic lower bounds in the cell-probe model. *SIAM Journal on Computing*, 35(4):932–963, 2006.

[53] A. Shamir. How to share a secret. *Communications of the ACM*, 22(11):612–613, 1979.

[54] A. C. Yao. How to generate and exchange secrets. In *Proceedings of FOCS*, pages 162–167, 1986.

[55] C. Zhu, S. Han, H. Mao, and W. J. Dally. Trained ternary quantization. *arXiv preprint arXiv:1612.01064*, 2016.

A Implementation Details

The complete CBT framework is available as a header-only C++ library at `https://github.com/queelius/cbt`. Key design decisions:

```cpp
template<typename T>
class cbt_name {
    static_assert(std::is_floating_point_v<T>, "...");
private:
    T internal_representation_;
public:
    // Factory methods for safe construction
    static cbt_name from_domain(T value);

    // Efficient operations in transformed domain
    cbt_name operator*(const cbt_name& other) const;

    // Inter-CBT mappings
    template<typename Other>
    Other to() const;
};
```

Listing 10: CBT design pattern

B Proofs of Theorems

B.1 Proof of Overflow-Free Mapping (Theorem 4)

We provide the deferred proof of Theorem 4 on overflow-free inter-CBT mappings.

Proof of Theorem 4. Let $\psi : D_1 \to D_2$ be a direct mapping between transformed domains with $\psi \circ \phi_1 = \phi_2$.

For any $x \in S_D$, the direct path computes:

$$x \xrightarrow{\phi_1} \phi_1(x) \xrightarrow{\psi} \psi(\phi_1(x)) = \phi_2(x)$$

Crucially, this computation never requires representing x or any intermediate value in D. The domain D may have constraints:

- **Range limits:** $S_D \subseteq [L, U]$ for some bounds L, U
- **Precision limits:** Representable values separated by gaps $\geq \epsilon$
- **Discrete structure:** S_D may be countable (e.g., machine integers)

The direct mapping ψ operates entirely within D_1 and D_2, which may have:

- Extended range: $S_{D_1}, S_{D_2} \supseteq \phi_1(S_D), \phi_2(S_D)$
- Different precision: No requirement to match D's precision
- Alternative structure: May be continuous where D is discrete

Therefore, overflow/underflow conditions in D cannot occur during the transformation. This is particularly valuable when D uses finite-precision arithmetic (e.g., IEEE 754) while D_1, D_2 use extended representations (e.g., logarithmic or multiscale domains). □ □

B.2 Additional Proofs

Lemma 1 (Homomorphism Preservation)**.** If $\phi : D_1 \to D_2$ is homomorphic for operation ω and $\psi : D_2 \to D_3$ is homomorphic for $\phi_*(\omega)$, then $\psi \circ \phi$ is homomorphic for ω.

Proof. By the homomorphic property:

$$
\begin{aligned}
(\psi \circ \phi)(\omega(x_1, \ldots, x_n)) &= \psi(\phi(\omega(x_1, \ldots, x_n))) && (37)\\
&= \psi(\phi_*(\omega)(\phi(x_1), \ldots, \phi(x_n))) && (38)\\
&= \psi_*(\phi_*(\omega))(\psi(\phi(x_1)), \ldots, \psi(\phi(x_n))) && (39)\\
&= (\psi \circ \phi)_*(\omega)((\psi \circ \phi)(x_1), \ldots, (\psi \circ \phi)(x_n)) && (40)
\end{aligned}
$$

Thus $(\psi \circ \phi)_*(\omega) = \psi_*(\phi_*(\omega))$ preserves the homomorphic property. □ □

Acknowledgments

The author thanks his Ph.D. advisor, Hiroshi Fujinoki, for his guidance and support. This work represents independent research conducted while pursuing doctoral studies at Southern Illinois University. The open-source implementation is available at `https://github.com/queelius/cbt`.

Operation	Baseline (mean ± SE)	CBT Implementation (mean ± SE)	Speedup	95% CI	p-value
Numerical Stability					
Product of 10^6 prob. ($p_i \sim 10^{-10}$)	Underflow at $n = 31$	Complete ($n = 10^6$)	–	–	–
Extreme range (10^{-600} to 10^{600})	Overflow/ underflow	Stable (multiscale)	–	–	–
Performance Improvements					
Bayesian update (1000 iterations)	847 ± 23 ms (w/ normalization)	12.0 ± 0.8 ms (odds ratio)	70.6×	[68.2, 73.1]	$< 10^{-8}$
1024-bit modular mult. (RSA operations)	3.20 ± 0.10 µs (Montgomery)	0.40 ± 0.02 µs (RNS, 8 moduli)	8.0×	[7.5, 8.6]	$< 10^{-8}$
Matrix convolution (512×512 FFT)	1847 ± 45 ms (direct $O(n^4)$)	234 ± 12 ms (FFT domain)	7.9×	[7.3, 8.5]	$< 10^{-8}$
Exactness Guarantees					
Rational arithmetic (10k ops, rel. error)	15.2 ± 0.4% (IEEE 754)	0.0 ± 0.0% (Stern-Brocot)	∞	–	–
Interval bounds (false positive rate)	47.3 ± 1.2% (point arith.)	0.0 ± 0.0% (interval CBT)	∞	–	–
Robustness					
3D rotation (singularities/1M)	3.1 ± 0.3 (Euler angles)	0.0 ± 0.0 (quaternions)	∞	–	–
AD gradient (100-var function)	342 ± 8 ms (finite diff.)	4.7 ± 0.2 ms (dual numbers)	72.8×	[69.8, 76.2]	$< 10^{-8}$

Table 3: Performance evaluation of CBT implementations. Results from $n = 1000$ trials per condition. Mean ± standard error shown. 95% confidence intervals computed via bootstrap (10,000 resamples). p-values from Welch's t-test with Bonferroni correction ($\alpha = 0.00625$ for 8 comparisons). Infinite speedup indicates elimination of errors/failures.

Operation	Theoretical Improvement	Measured Speedup	Cohen's d (Effect Size)	Interpretation
Bayesian update	$O(n) \to O(1)$	70.6×	12.3	Very large
RNS multiplication	$O(\log n) \to O(1)$	8.0×	8.7	Very large
FFT convolution	$O(n^2) \to O(n \log n)$	7.9×	9.2	Very large
AD gradient	$O(n^2) \to O(n)$	72.8×	14.1	Very large

Table 4: Agreement between theoretical predictions and empirical measurements

Transform	Improved Op.	Degraded Op.	Degradation Factor
Logarithmic	Multiplication	Addition	12.3× slower
RNS	Multiplication	Comparison	1830× slower
Odds-ratio	Bayes update	Marginalization	3.2× slower
Quaternion	Rotation comp.	Euler extraction	4.1× slower

Table 5: Trade-offs observed in CBT implementations

Transform	Base Size	CBT Size	Overhead	Justification
IEEE double	8 bytes	8 bytes	0%	Baseline
Logarithmic	8 bytes	8 bytes	0%	Same representation
Odds-ratio	8 bytes	8 bytes	0%	Single value stored
Dual numbers	8 bytes	16 bytes	100%	Value + derivative
Interval	8 bytes	16 bytes	100%	Lower + upper bound
Multiscale	8 bytes	16 bytes	100%	Mantissa + scale
Quaternion	12 bytes	16 bytes	33%	4D vs 3D representation
Stern-Brocot	8 bytes	24 bytes	200%	Path in tree stored
RNS (8 moduli)	8 bytes	64 bytes	700%	8 residues stored

Table 6: Memory overhead of CBT representations. Overhead is acceptable when amortized over performance gains.

Task	Specialized Library	CBT	Speedup	Notes
Arbitrary precision	MPFR (256-bit)	Multiscale	3.2×	Limited precision but faster
Exact rationals	GMP rationals	Stern-Brocot	1.8×	Tree navigation vs arbitrary
Interval arithmetic	MPFI	Interval CBT	1.1×	Comparable performance
Automatic diff.	ADOL-C	Dual numbers	2.4×	Forward mode only
Modular arithmetic	OpenSSL	RNS	8.0×	For parallel operations
3D rotations	Eigen	Quaternions	0.95×	Slightly slower, no singularities

Table 7: CBT performance versus specialized libraries. CBTs excel when operations match the transform's strengths.

Scenario	Standard	CBT	Reason for Failure
Single operation	10 ns	50 ns	Transformation overhead dominates
Random access pattern	100 ms	850 ms	Poor cache locality in transformed domain
Mixed operations	200 ms	580 ms	Frequent domain switching
Small data ($n < 10$)	15 ns	120 ns	Overhead not amortized
General computation	Baseline	2-10× slower	No dominant operation to optimize

Table 8: Failure cases where CBTs underperform. CBTs require sufficient operation repetition to amortize transformation costs.

Approach	**Focus**	**How CBT Differs**	**CBT Advantage**
Program Synthesis [29]	Generate programs from specs	CBT transforms representations, not programs	Systematic trade-off analysis
Compiler Optimization	Improve code within fixed representation	CBT changes the representation itself	Fundamental complexity changes
Data Structures [52]	Trade space for time	CBT trades operation complexity	Broader applicability
Monads [19]	Manage computational effects	CBT focuses on performance	Quantitative trade-offs
Algorithm Selection [**?**]	Choose best algorithm	CBT transforms to enable algorithms	Representation-aware
Homomorphic Encryption [10]	Compute on encrypted data	Special case of CBT with security focus	General framework
Automatic Differentiation	Compute derivatives	AD is one CBT instance	Unified with other transforms
Number Systems	Alternative representations	Individual techniques	Systematic framework

Table 9: Comparison of CBT with related approaches. CBT provides a unifying lens for understanding diverse transformation techniques.