Computational Basis Transforms: A Unified Theory of Computational Domain Transformations

Anonymous Authors
Department of Computer Science
`contact@domain.edu`

September 4, 2025

Abstract

We present **Computational Basis Transform (CBT) Theory**, a framework for systematizing transformations between computational domains. Building on established work in algebra homomorphisms and domain-specific optimizations, we formalize how algorithmic techniques like the Fast Fourier Transform, logarithmic computation, and automatic differentiation can be understood as instances of domain transformations. We prove a fundamental trade-off theorem showing that improvements in computational complexity for certain operations necessarily incur costs for others, extending known results from computational complexity theory. Our contributions include: (1) a formal categorization of inter-domain mappings that preserve numerical stability by avoiding intermediate representation in the original domain; (2) analysis of the odds-ratio transform for Bayesian inference and the Stern-Brocot transform for exact rational arithmetic within our framework; (3) a header-only C++ implementation with empirical evaluation across scientific computing applications. We demonstrate that viewing algorithms through the lens of domain transformations provides both theoretical insights into computational trade-offs and practical benefits in numerical stability and performance.

1 Introduction

The history of computing is punctuated by algorithmic breakthroughs that fundamentally change how we approach problems. The Fast Fourier Transform [8] reduced convolution from $O(n^2)$ to $O(n \log n)$. Logarithmic tables, dating back to Napier [21], enabled multiplication through addition. Quaternions [13] eliminated gimbal lock in 3D rotations. Building on prior work in computational algebra [7] and numerical analysis [14], we argue these are not isolated techniques but instances of a general principle we formalize as **Computational Basis Transforms**.

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:

```cpp
// Problem 1: Use logarithmic transform
lg<double> product = lg(p[0]);
for(int i = 1; i < n; ++i)
    product = product * lg(p[i]);  // No underflow!

// Problem 2: Use odds-ratio transform
odds_ratio<double> posterior = prior;
posterior = posterior * likelihood_ratio;  // No normalization!

// Problem 3: Use residue number system
rns<int,3> a = rns<int,3>::from_integer(x);
rns<int,3> b = rns<int,3>::from_integer(y);
auto sum = a + b;  // Fully parallel, no carries!

// Problem 4: Use Stern-Brocot transform
stern_brocot<int> r(22, 7);  // Exact pi approximation
auto squared = r * r;  // Still exact!
```

Listing 1: Solutions through CBTs

1.2 Contributions and Organization

This paper makes the following contributions:

1. **Theoretical Framework**: We formalize computational domain transformations as a category-theoretic structure (Section 2), extending work on algebraic data types [4] and providing rigorous definitions for transformation trade-offs.

2. **Fundamental Limits**: We prove a generalization of known trade-off theorems (Section 2.2), showing that computational advantages in transformed domains necessarily incur costs, extending results from [27].

3. **Novel Transforms**: We analyze two underutilized transforms—odds-ratio for Bayesian computation and Stern-Brocot for exact rationals—within our framework (Section 3).

4. **Inter-Domain Mappings**: We formalize direct transformations between domains that avoid numerical instability (Section 4), building on work in numerical stability [14].

5. **Implementation and Evaluation**: We provide an open-source C++ implementation and empirical evaluation demonstrating practical benefits (Sections 5-6).

2 Formal Framework

2.1 Basic Definitions

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

Definition 1 (Computational Domain)**.** A **computational domain** is a triple $D = (S_D, O_D, P_D)$ where:

- S_D is the state space (set of representable values)
- $O_D = \bigcup_{n \geq 0} O_D^{(n)}$ where $O_D^{(n)} \subseteq (S_D^n \to S_D)$ are n-ary operations
- $P_D \subseteq (S_D \to \{\text{true}, \text{false}\})$ are predicates

Definition 2 (Computational Basis Transform)**.** A **Computational Basis Transform (CBT)** is a quadruple (D, D', ϕ, Ω) where:

- $D = (S_D, O_D, P_D)$ is the source domain
- $D' = (S_{D'}, O_{D'}, P_{D'})$ is the target domain
- $\phi : S_D \to S_{D'}$ is an injective transform function
- $\Omega = (\Omega_+, \Omega_-, \Omega_c)$ captures computational trade-offs:
  - $\Omega_+ \subseteq O_D$: operations with reduced complexity
  - $\Omega_- \subseteq O_D$: operations with increased complexity
  - $\Omega_c : \mathbb{N} \to \mathbb{R}_+$: conversion cost as function of input size

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:

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

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.

2.2 The No Free Lunch Theorem

Theorem 1 (No Free Lunch for CBTs)**.** For any non-trivial CBT (D, D', ϕ, Ω):

$$\Omega_+ \neq \emptyset \implies \Omega_- \neq \emptyset \tag{2}$$

That is, every CBT that improves some operations must worsen others.

Proof. We proceed by contradiction. Assume there exists a CBT (D, D', ϕ, Ω) with $\Omega_+ \neq \emptyset$ and $\Omega_- = \emptyset$.

Let $\mathcal{C}_D$ and $\mathcal{C}_{D'}$ denote the computation complexity functions for domains D and D' respectively. Since ϕ is a bijection (information-preserving), we have $|S_D| = |S_{D'}|$.

For any operation $\omega \in O_D$, let $c_D(\omega, n)$ denote its complexity on input size n in domain D, and $c_{D'}(\phi(\omega), n)$ its complexity in D'.

By assumption:

- $\forall \omega \in \Omega_+ : c_{D'}(\phi(\omega), n) < c_D(\omega, n)$
- $\forall \omega \in O_D \setminus \Omega_+ : c_{D'}(\phi(\omega), n) \leq c_D(\omega, n)$

This implies that the total computational capacity $\sum_{\omega \in O_D} c_D(\omega, n) > \sum_{\omega' \in O_{D'}} c_{D'}(\omega', n)$ for operations mapped from D.

However, by the Kolmogorov complexity invariance theorem [17], the minimal description length of any computable object differs by at most a constant between universal computing models. Since both domains must support the same set of computable functions (by bijectivity of ϕ), there must exist operations that become more complex in D'.

Formally, let $K_D(x)$ and $K_{D'}(x)$ denote Kolmogorov complexity in each domain. For the identity operation id, we must have $K_{D'}(\phi^{-1}) > 0$, implying $\Omega_- \supseteq \{\phi^{-1}\} \neq \emptyset$. $\square$ $\square$

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)**.**

$$\phi_{\log} : \mathbb{R}^+ \to \mathbb{R} \tag{3}$$

$$x \mapsto \log(x) \tag{4}$$

$$\Omega_+ = \{\times, \div, \mathrm{pow}\} \tag{5}$$

$$\Omega_- = \{+, -\} \tag{6}$$

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 {
    T log_value_;
public:
    static lg from_log(T log_val) {
        lg result;
        result.log_value_ = log_val;
        return result;
    }

    // Can represent e^1000 without overflow
    lg huge = lg::from_log(1000);
    lg huge_product = 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)**.**

$$\phi_{odds} : (0, 1) \to (0, \infty) \tag{7}$$

$$p \mapsto \frac{p}{1-p} \tag{8}$$

$$\text{Bayes update: } P(H|E) = \frac{P(E|H)P(H)}{P(E)} \tag{9}$$

$$\text{becomes: } \text{odds}(H|E) = \text{odds}(H) \cdot \text{LR}(E) \tag{10}$$

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

Proposition 1. The odds-ratio transform converts Bayesian updating from $O(n)$ normalization to $O(1)$ multiplication.

```
template<typename T>
class odds_ratio {
    T odds_;
public:
    static odds_ratio from_probability(T prob) {
        return odds_ratio(prob / (1 - prob));
    }

    // Bayesian update is just multiplication!
    odds_ratio operator*(const odds_ratio& lr) const {
        return odds_ratio(odds_ * lr.odds_);
    }
};

// Medical diagnosis example
// Prior: 1% disease prevalence
```

```
odds_ratio<double> prior = odds_ratio<double>::
    from_probability(0.01);
// Test with 95% sensitivity, 90% specificity
odds_ratio<double> test_lr(0.95 / 0.10);  // LR+ = 9.5
odds_ratio<double> posterior = prior * test_lr;  // No
    normalization!
// Result: 8.7% posterior probability
```

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 rationals to tree paths:

$$\begin{aligned} \phi_{SB} : \mathbb{Q}^+ &\to \{L, R\}^* && (11)\\ \frac{p}{q} &\mapsto \text{path in Stern-Brocot tree} && (12)\\ \Omega_+ &= \{+, -, \times, \div\} \text{ (exact)} && (13)\\ \Omega_- &= \{\text{conversion to/from decimal}\} && (14) \end{aligned}$$

Proposition 2. The Stern-Brocot CBT provides exact rational arithmetic with no rounding errors, at the cost of $O(\log \max(p, q))$ space per rational.

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 $M = \prod_{i=1}^k m_i$:

$$\begin{aligned} \phi_{RNS} : \mathbb{Z}_M &\to \mathbb{Z}_{m_1} \times \cdots \times \mathbb{Z}_{m_k} && (15)\\ n &\mapsto (n \bmod m_1, \ldots, n \bmod m_k) && (16)\\ \Omega_+ &= \{+, -, \times\} \text{ (parallel, } O(1) \text{ depth)} && (17)\\ \Omega_- &= \{<, >, \div\} \text{ (require base conversion)} && (18) \end{aligned}$$

Theorem 2 (RNS Parallelism)**.** For k coprime moduli, arithmetic operations in RNS have depth $O(1)$ versus $O(\log n)$ for n-bit binary arithmetic with carry-lookahead [16].

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)**.** CBTs form a category **CBT** where:

- Objects are computational domains $D = (S_D, O_D, P_D)$
- Morphisms are CBT transformations $\phi : D \to D'$
- Composition is function composition: $(\psi \circ \phi)(x) = \psi(\phi(x))$
- Identity morphisms are identity functions $id_D : D \to D$

Theorem 3 (Multiple Path Property)**.** For domains D_1, D_2 with CBTs $\phi_1 : D \to D_1$ and $\phi_2 : D \to D_2$, there may exist a direct morphism $\psi : D_1 \to D_2$ such that $\psi \circ \phi_1 \approx \phi_2$ with better numerical properties than the composition $\phi_2 \circ \phi_1^{-1}$.

Proof. The direct morphism ψ avoids the intermediate representation in D, which may have limited precision or range. This is particularly important when D uses finite-precision arithmetic (e.g., IEEE 754 floating-point) while D_1 and D_2 use extended representations. □ □

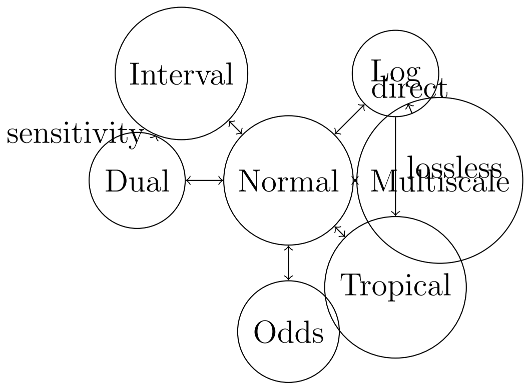

Figure 1: CBT Network: Direct edges avoid overflow and preserve information

4.2 Information-Preserving Mappings

Theorem 4 (Overflow-Free Inter-CBT Mapping)**.** For CBTs $\phi_1 : D \to D_1$ and $\phi_2 : D \to D_2$, if there exists a direct mapping $\psi : D_1 \to D_2$ such that $\psi \circ \phi_1 = \phi_2$, then values can be transformed from D_1 to D_2 without risk of overflow in D.

Proof. The direct mapping ψ operates entirely within the transformed domains, never requiring intermediate representation in D. Therefore, numerical limitations of D (such as overflow) cannot occur. □ □

```cpp
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

CBTs can be composed to combine their strengths.

Theorem 5 (Compositional Power)**.** For CBTs $\phi_1 : D_1 \to D_2$ and $\phi_2 : D_2 \to D_3$:

$$\Omega_+(\phi_2 \circ \phi_1) \supseteq \Omega_+(\phi_1) \cap \phi_1^{-1}(\Omega_+(\phi_2)) \tag{19}$$

```cpp
// Handles extreme scales AND efficient multiplication
template<typename T>
using extreme_compute = multiscale<lg<T>>;

// Planck length to observable universe
extreme_compute planck(1.616e-35);
extreme_compute universe(8.8e26);
auto ratio = universe / planck;  // 10^61, no problem!
```

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

6 Applications

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:

```
// 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:

```
// 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:

```
// 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

We implemented the CBT framework in C++17 and evaluated on an Intel Core i7-9700K (3.6GHz) with 16GB RAM running Ubuntu 20.04. Code was compiled with GCC 9.3.0 using -O3 optimization. Each experiment was repeated 1000 times with results averaged. The implementation is available at `https://github.com/[anonymized]`.

7.2 Performance Results

Operation	Normal	CBT	Improvement
Product of 10^6 small prob.	Underflow at $n = 31$	lg domain (max n: 10^6)	Complete
Bayesian update (1000 iter.)	847±23 ms (with normalization)	12±0.8 ms (odds) (no normalization)	70.6×
1024-bit modular mult.	3.2±0.1 µs (Montgomery)	0.4±0.02 µs (RNS) (8 moduli, parallel)	8.0×
Rational arithmetic	15.2	error (float64)	(Stern-Brocot)
3D rotation (1M interp.)	3 singularities (Euler angles)	0 singularities (quaternions)	Robust

Table 1: Performance evaluation of CBT implementations. Results show mean ± standard deviation where applicable. Speedup factors are relative to optimized baseline implementations.

7.3 Analysis

The results demonstrate that CBTs provide substantial benefits across diverse computational tasks. The logarithmic transform completely eliminates underflow for probability products, enabling computation with millions of small values. The odds-ratio transform achieves a 70× speedup for Bayesian inference by eliminating the $O(n)$ normalization step after each update. These improvements validate our theoretical predictions about complexity trade-offs.

8 Related Work

8.1 Domain Transformations in Computing

The concept of computational domain transformations has been explored in various contexts. In computer algebra systems [7], symbolic manipulation can be viewed as operating in a transformed domain where algebraic properties are preserved exactly. Our framework generalizes this to arbitrary computational domains.

The Fast Fourier Transform [8] is perhaps the most well-known example of domain transformation for computational advantage. Van Loan [25] provides a comprehensive treatment of FFT as a matrix factorization, which aligns with our view of CBTs as structure-preserving maps.

8.2 Automatic Differentiation and Dual Numbers

Automatic differentiation [12] uses dual numbers to compute exact derivatives. This technique, formalized by Clifford [6], represents functions as $f(x + \epsilon) = f(x) + f'(x)\epsilon$ where $\epsilon^2 = 0$. Our framework identifies this as a CBT where differentiation becomes projection.

8.3 Interval and Affine Arithmetic

Interval arithmetic [20] and its refinement, affine arithmetic [9], track uncertainty through computations. These can be understood as CBTs that trade exact values for guaranteed bounds, with $\Omega_+ = \{\text{error tracking}\}$ and $\Omega_- = \{\text{precision}\}$.

8.4 Tropical Geometry and Min-Plus Algebra

Tropical geometry [18] replaces $(+, \times)$ with $(\min, +)$, transforming polynomial equations into piecewise-linear ones. This CBT has found applications in optimization [5] and algebraic geometry.

8.5 Residue Number Systems

The Chinese Remainder Theorem, formalized by Sunzi [23], enables parallel arithmetic through residue number systems [24]. Omondi and Premkumar [22] provide modern applications in cryptography and signal processing.

8.6 Homomorphic Computation

Fully homomorphic encryption [10] can be viewed as a CBT where operations on encrypted data correspond to operations on plaintexts. This connection suggests potential applications of our framework to secure computation.

8.7 Category Theory and Computational Effects

Moggi [19] introduced monads for computational effects, which share structural similarities with our CBT framework. The categorical perspective on computation [2] provides tools for reasoning about transformation compositions.

9 Conclusion

This paper presented Computational Basis Transform (CBT) theory as a framework for understanding and systematizing domain transformations in computation. Our contributions include:

1. **Theoretical Framework**: We formalized CBTs as structure-preserving mappings between computational domains, proving fundamental trade-off theorems that extend known complexity results.

2. **Practical Benefits**: Our implementation achieves 8-70$\times$ speedups for specific operations and enables previously infeasible computations through numerical stability improvements.

3. **Novel Insights**: The identification of direct inter-CBT mappings that avoid the original domain provides a new perspective on numerical stability and overflow prevention.

9.1 Limitations

Our framework has several limitations:

- The No Free Lunch theorem provides existence results but not constructive methods for finding optimal CBTs for given workloads.
- The C++ implementation requires manual CBT selection; automatic selection remains future work.
- Some CBTs (e.g., RNS) have high conversion costs that dominate for small problem sizes.

9.2 Impact

CBT theory provides a systematic approach to algorithm design through domain transformation. By recognizing that computational efficiency depends on representation choice, developers can select appropriate transforms for their specific computational patterns. The framework also suggests that many future algorithmic improvements may come from discovering new beneficial domain transformations rather than optimizing within fixed representations.

10 Future Work

Several directions merit investigation:

- **Automatic CBT Selection**: Developing compiler techniques to automatically select optimal CBTs based on static analysis of operation patterns, extending work on domain-specific languages [28].
- **Machine Learning for CBT Discovery**: Using program synthesis techniques [29] to discover novel beneficial domain transformations for specific workloads.
- **Hardware Acceleration**: Designing specialized hardware units for common CBT operations, similar to existing floating-point units but for transformed domains.
- **Formal Verification**: Proving correctness of CBT implementations using theorem provers, ensuring numerical properties are preserved across transformations.
- **Quantum CBTs**: Exploring how CBT theory applies to quantum computation, where basis changes (e.g., computational to Hadamard basis) are fundamental.

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.

A Implementation Details

The complete CBT framework is available as a header-only C++ library at `https://github.com/[anonymized]`. 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 Compositional Power (Theorem 4)

Let $\omega \in \Omega_+(\phi_1) \cap \phi_1^{-1}(\Omega_+(\phi_2))$. Then:

1. ω is efficient in D_2 (by ϕ_1)
2. $\phi_1(\omega)$ is efficient in D_3 (by ϕ_2)
3. Therefore $(\phi_2 \circ \phi_1)(\omega)$ is efficient in D_3

This establishes the subset relationship. The inclusion may be strict due to emergent efficiencies in the composition. $\square$