A Turing-Complete Resistance-Based Isomorphism

for Probabilistic Computation

Alexander Pisani

a.h.pisani.research@gmail.com

December 2025

Submitted for Review

Abstract

This paper establishes a rigorous isomorphism between probability mathematics and informational resistance. The fundamental translation $\Omega(P) = -\ln(P)$ maps probability to resistance, transforming multiplication into addition. This correspondence extends across four equivalent representations (natural language, probability theory, circuit topology, and prime coordinate vectors) which function as a Rosetta Stone for discrete mathematics: the same computation can be expressed in any layer and translated exactly to the others.

Boolean logic emerges from circuit topology: AND as series (resistances add), OR as parallel (conductances add), and NOT as phase interference. We prove the framework's validity through two complete worked examples. First, we derive the probability that two random integers are coprime, obtaining the known result $6/\pi^2$ by traversing all four representation layers. Second, we demonstrate that relational database operations (JOIN, UNION, INTERSECTION) map directly to number-theoretic operations (GCD, LCM). We establish Turing completeness by constructing an explicit simulation of counter machines using prime exponents as registers.

The framework reveals that prime numbers are precisely the irreducible elements of this informational structure, configurations whose resistance cannot be decomposed into sums of smaller resistances. This characterization, combined with the Gödel-style encoding of data through prime coordinates, suggests that any sufficiently powerful computational system must rediscover the primes as a structural necessity.

PART I: FOUNDATIONS

1. Introduction

The prime numbers have fascinated mathematicians for millennia. They appear to be fundamental building blocks of arithmetic, yet their distribution along the number line exhibits patterns that remain mysterious despite centuries of study. This paper takes a novel approach: rather than treating primes as given mathematical objects, we develop a computational framework in which primes are characterized as the irreducible elements of an informational structure.

Our central insight is that the mathematical structure of probability multiplication maps precisely onto the physical behavior of resistors in series. When two independent events must both occur (logical AND), their probabilities multiply:

$$\mathrm{P}(\mathrm{A} \cap \mathrm{B}) = \mathrm{P}(\mathrm{A}) \times \mathrm{P}(\mathrm{B})$$

In circuit terms, resistors in series add:

$$\Omega\_\mathrm{total} = \Omega\_\mathrm{A} + \Omega\_\mathrm{B}$$

The logarithm mediates between these domains, transforming multiplication into addition. This is not merely an analogy; it is an exact mathematical isomorphism that preserves algebraic structure.

1.1 The Four-Layer Framework

We demonstrate that every computation in this framework can be expressed equivalently in four representations:

1. English: Natural language description of the computation
2. Probability: Mathematical formulation using probability theory
3. Circuit: Physical topology of resistors and conductances
4. Vector Space: Coordinates in infinite-dimensional prime space

Each layer provides different intuitions and computational tools, but all four are mathematically equivalent. A proof in one layer automatically translates to proofs in the others.

2. The Fundamental Mapping

2.1 Informational Resistance

Definition 2.1 (Informational Resistance). For a probability value P ∈ (0, 1], the informational resistance is:

$$\Omega(\mathrm{P}) = -\ln(\mathrm{P})$$

For P = 0, we define Ω(0) = ∞. This establishes a bijection between (0, 1] and [0, ∞), mapping certainty (P = 1) to zero resistance and impossibility (P → 0) to infinite resistance.

The connection to information theory is immediate: −ln(P) is precisely the self-information (surprisal) of an event with probability P, measured in nats. Thus informational resistance quantifies the "surprise" or "information cost" associated with an event occurring.

2.2 The Binary Extremes

State 1 (True/Certain): Ω(1) = −ln(1) = 0. Zero resistance; information flows unimpeded.

State 0 (False/Impossible): Ω(0) = −ln(0) = ∞. Infinite resistance; the circuit is broken.

These boundary conditions ensure that classical binary logic emerges naturally from the continuous probability framework.

2.3 The Isomorphism Theorem

Theorem 2.2 (Probability-Resistance Homomorphism). The map Ω: (0, 1] → [0, ∞) defined by Ω(P) = −ln(P) is a monoid isomorphism from ((0, 1], ×) to ([0, ∞), +). That is:

$$\Omega(\mathrm{P}_1 \times \mathrm{P}_2) = \Omega(\mathrm{P}_1) + \Omega(\mathrm{P}_2)$$

Proof. Direct application of the logarithm property: $\Omega(P_1 \times P_2) = -\ln(P_1 \times P_2) = -\ln(P_1) - \ln(P_2) = \Omega(P_1) + \Omega(P_2)$. The map is bijective with inverse P = e^(−Ω), and both Ω and its inverse are continuous. □

Note: We say monoid rather than group because ((0, 1], ×) lacks multiplicative inverses within the interval (e.g., the inverse of 0.5 is 2, which lies outside (0, 1]). However, the map extends naturally to a full group isomorphism from ((0, ∞), ×) to ($\mathbb{R}$, +), corresponding to the extension from probabilities to likelihood ratios.

2.4 Uniqueness

Theorem 2.3. The mapping Ω(P) = −ln(P) is the unique continuous function f: (0, 1] → [0, ∞) satisfying $f(P_1 \cdot P_2) = f(P_1) + f(P_2)$, up to a positive multiplicative constant.

Proof sketch. The functional equation f(xy) = f(x) + f(y) is Cauchy's logarithmic equation. For continuous functions, the only solutions are f(x) = c·ln(x) for some constant c. The constraint f: (0, 1] → [0, ∞) requires c < 0; normalizing so that f(1/e) = 1 gives c = −1, yielding f(P) = −ln(P). See Aczél (1966), Lectures on Functional Equations, for the complete treatment. □

This theorem establishes that our framework is not arbitrary: the logarithmic mapping is forced by the requirement that series combination be additive. Any other continuous mapping would fail to preserve the algebraic structure.

PART II: BOOLEAN LOGIC

3. AND Logic: Series Circuits

Theorem 3.1 (AND-Series Correspondence). For independent events A and B, the AND operation P(A ∩ B) = P(A) × P(B) corresponds to series circuit addition:

$$\Omega(\mathrm{A} \cap \mathrm{B}) = \Omega(\mathrm{A}) + \Omega(\mathrm{B})$$

3.1 Circuit Topology

In a series circuit, current must pass through both resistors. The total resistance is the sum of individual resistances. This directly implements AND logic: both conditions must be satisfied.

SERIES CIRCUIT (AND GATE)

$$\Omega\_\mathrm{total} = \Omega\_\mathrm{A} + \Omega\_\mathrm{B} \quad \Longrightarrow \quad \mathrm{P\_total} = \mathrm{P\_A} \times \mathrm{P\_B}$$

3.2 Worked Example: Drawing a Red Ace

English: "What is the probability of drawing a Red Ace from a standard deck of cards?"

This requires both conditions to be true: the card must be Red AND the card must be an Ace. These attributes are independent: knowing a card is Red does not change the probability it is an Ace (2 of 26 red cards are Aces = 1/13, same as 4 of 52 total). Thus P(Ace|Red) = P(Ace), and we can multiply.

Probability Formulation:

- P(Ace) = 4/52 = 1/13
- P(Red) = 26/52 = 1/2
- P(Red Ace) = P(Ace) × P(Red) = 1/13 × 1/2 = 1/26

Circuit Formulation:

- Ω(Ace) = −ln(1/13) = ln(13) ≈ 2.565
- Ω(Red) = −ln(1/2) = ln(2) ≈ 0.693
- Ω(Red Ace) = Ω(Ace) + Ω(Red) = 2.565 + 0.693 = 3.258

Verification: e^(−3.258) = 1/26 ≈ 0.0385 ✓

4. OR Logic: Parallel Circuits

4.1 Mutually Exclusive Events

Theorem 4.1 (OR-Parallel Correspondence, Exclusive Case). For mutually exclusive events A and B (where P(A ∩ B) = 0), the OR operation corresponds to parallel circuit conductance addition:

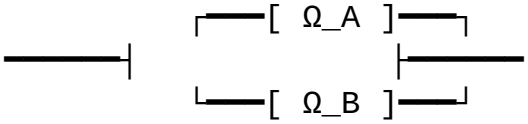
$$\mathrm{G\_total} = \mathrm{G(A)} + \mathrm{G(B)} = \mathrm{P(A)} + \mathrm{P(B)}$$

where G = P in the informational domain.

4.2 Circuit Topology

In a parallel circuit, current can flow through either path. The total conductance is the sum of individual conductances. This implements OR logic: either condition being satisfied allows flow.

PARALLEL CIRCUIT (OR GATE)



$$\mathrm{G\_total} = \mathrm{G\_A} + \mathrm{G\_B} \quad \Longrightarrow \quad \mathrm{P\_total} = \mathrm{P\_A} + \mathrm{P\_B}$$

4.3 Worked Example: Rolling a 2 or 3

English: "What is the probability of rolling a 2 OR a 3 on a fair six-sided die?"

These are mutually exclusive events: you cannot roll both a 2 and a 3 simultaneously.

Probability Formulation:

- P(roll 2) = 1/6, P(roll 3) = 1/6
- P(roll 2 OR roll 3) = 1/6 + 1/6 = 2/6 = 1/3

Circuit Formulation:

- G(roll 2) = 1/6, G(roll 3) = 1/6
- G_total = 1/6 + 1/6 = 1/3
- Ω_total = −ln(1/3) = ln(3) ≈ 1.099

Verification: P(roll 2 OR roll 3) = 2/6 = 1/3 ✓

4.4 Inclusive OR: Handling Non-Exclusive Events

For events that can co-occur (P(A ∩ B) ≠ 0), the standard inclusion-exclusion principle applies:

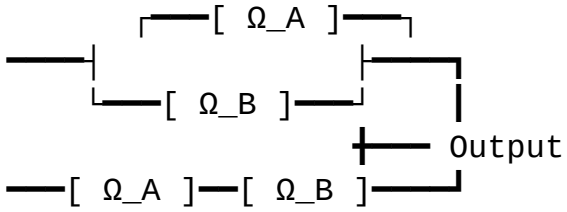
$$\mathrm{P(A \cup B) = P(A) + P(B) - P(A \cap B)}$$

In circuit terms, this requires a correction for the "overlap conductance":

$$\mathrm{G\_inclusive = G\_A + G\_B - G\_\{A \cap B\}}$$

The correction term G_{A∩B} = P(A ∩ B) is computed via AND (series circuit), then subtracted using our phase-shift mechanism. The circuit topology becomes:

INCLUSIVE OR CIRCUIT



(π-shifted: subtracts overlap)

The lower branch computes P(A ∩ B) via series, then phase-shifts by π to produce −P(A ∩ B). Summing with the parallel branch yields P(A) + P(B) − P(A ∩ B).

4.5 The Parallel Resistance Formula

For n parallel paths with resistances Ω_1, Ω_2, ..., Ω_n:

$$\mathrm{\Omega\_total = -\ln(e\hat{}(-\Omega_1) + e\hat{}(-\Omega_2) + ... + e\hat{}(-\Omega_n))}$$

This log-sum-exp formula appears in statistical mechanics (partition functions), machine learning (softmax normalization), and information theory, revealing deep structural connections across disciplines.

5. NOT Logic: The Phase-Shift Inverter

5.1 The Challenge

To complete universal Boolean logic, we require a NOT operation: NOT(A) means P(¬A) = 1 − P(A). This presents a fundamental challenge: our framework is built on multiplication (AND) and addition (OR), but NOT requires subtraction from unity.

In the resistance domain, subtraction has no natural interpretation: resistances only add in series. To enable NOT, we must extend the framework from real-valued probabilities to complex amplitudes, analogous to the extension from DC circuits (pure resistance) to AC circuits (impedance with phase).

5.2 The Complex Amplitude Framework

We represent probability states in polar form:

$$Z = P \cdot e^{\wedge}(i\theta) = P(\cos\,\theta + i\,\sin\,\theta)$$

where P = |Z| is the probability magnitude and θ is the phase angle.

What phase represents: Phase is a bookkeeping degree of freedom that enables interference. Just as quantum mechanical amplitudes carry phase that cancels in destructive interference, our complex probabilities carry phase that enables subtraction through the identity e^(iπ) = −1.

Interaction with other operations:

- AND (multiplication): Phases add. $Z_1 \times Z_2 = P_1P_2 \cdot e^{\wedge}(i(\theta_1+\theta_2))$
- OR (addition): Phases must match for coherent addition, or be tracked separately
- NOT (interference): Phase enables subtraction via π-shift

Observable output: Only the magnitude |Z| = P is physically meaningful as probability. Phase is internal to the computation, analogous to quantum amplitudes where only $|\psi|^2$ is observable. This is why the NOT operation preserves phase: it tracks identity through the inversion.

5.3 The Phase-Shift Inverter

We implement NOT using destructive interference. The key insight is that multiplying by e^(iπ) = −1 produces a 180° phase shift, enabling subtraction.

Definition 5.1 (NOT Operation). For a complex probability state Z = P·e^(iθ):

$$\mathrm{NOT}(Z) = (Z/|Z|) - Z = (1 - |Z|) \cdot e^{\wedge}(i\cdot\arg(Z))$$

The operation requires a reference signal at P = 1 (certainty) to compute the complement. This is analogous to needing a voltage reference in electronic inverters. The three steps:

1. Normalize: Extract phase to create reference Z_ref = e^(iθ) (magnitude 1, same phase)
2. π-Shift: Multiply input by −1 to produce Z' = −P·e^(iθ)
3. Sum: Add reference and shifted input: Z_out = e^(iθ) − P·e^(iθ) = (1−P)·e^(iθ)

5.4 Verification

Test case: NOT(0.3) with phase θ = π/4

Step	Calculation	Result
Input	Z_in = 0.3·e^(iπ/4)	0.212 + 0.212i
Normalize	Z_ref = e^(iπ/4)	0.707 + 0.707i
π-shift	Z' = −Z_in	−0.212 − 0.212i
Sum	Z_out = Z_ref + Z'	0.495 + 0.495i
Magnitude	|Z_out|	**0.7** ✓
Phase	arg(Z_out)	**π/4 (preserved)** ✓

Double inversion test: NOT(NOT(0.3)) = NOT(0.7) = 0.3 ✓

6. Complete Boolean Logic

With AND, OR, and NOT gates established, the framework possesses a functionally complete set of Boolean operators. Any Boolean function can be constructed from these primitives.

6.1 Derived Gates

Gate	Construction	Property
NAND	NOT(AND(A, B))	Universal
NOR	NOT(OR(A, B))	Universal
XOR	(A AND NOT B) OR (NOT A AND B)	Exclusive or
XNOR	NOT(XOR(A, B))	Equivalence

Note: XOR is implemented as the OR of two AND branches, each combining one input with the NOT of the other. For independent events A and B, the probability formula P(A XOR B) = P(A)(1−P(B)) + (1−P(A))P(B) = P(A) + P(B) − 2P(A)P(B) follows from expanding this construction. The independence assumption is required for the multiplicative decomposition P(A ∧ ¬B) = P(A)P(¬B). In circuit terms, XOR requires series (AND), phase-shift (NOT), and parallel (OR) operations combined.

6.2 Binary Truth Table Verification

We verify AND and NOT using boundary values P = 1 (TRUE) and P = 0 (FALSE):

AND Gate:

A	B	Ω_A	Ω_B	P_out
1	1	0	0	**1** ✓
1	0	0	∞	**0** ✓
0	1	∞	0	**0** ✓
0	0	∞	∞	**0** ✓

PART III: THE VECTOR SPACE LAYER

7. Prime Coordinates

7.1 The Fundamental Theorem of Arithmetic

Every positive integer n > 1 has a unique representation as a product of prime powers:

$$n = 2^{\wedge}(a_2) \times 3^{\wedge}(a_3) \times 5^{\wedge}(a_5) \times 7^{\wedge}(a_7) \times ...$$

The exponents (a_2, a_3, a_5, a_7, ...) form an infinite-dimensional coordinate vector:

$$v(n) = (a_2, a_3, a_5, a_7, ...)$$

7.2 Examples

n	Factorization	Coordinate Vector
12	$2^2 \times 3^1$	(2, 1, 0, 0, ...)
35	$5^1 \times 7^1$	(0, 0, 1, 1, 0, ...)
2	2^1 (prime)	(1, 0, 0, 0, ...), single non-zero
30	$2^1 \times 3^1 \times 5^1$	(1, 1, 1, 0, ...)

Key insight: Primes have exactly one non-zero coordinate. This is their defining characteristic in the vector space.

Identity element: The multiplicative identity 1 has no prime factors, so v(1) = (0, 0, 0, ...), the zero vector. This is consistent: multiplying by 1 leaves any number unchanged, and adding the zero vector leaves any coordinate unchanged.

7.3 Linear Independence

Theorem 7.1. The set {ln(p) : p prime} is linearly independent over the rationals.

Proof. Suppose there exists a non-trivial rational linear combination Σ q_p · ln(p) = 0 with not all q_p zero. Since any such combination involves only finitely many terms with non-zero coefficients, we may clear denominators to obtain integer coefficients: Σ a_p · ln(p) = 0. Exponentiating both sides: ∏ p^(a_p) = 1, which is a well-defined positive rational. But by the Fundamental Theorem of Arithmetic, the only product of prime powers equaling 1 has all exponents zero. This contradicts our assumption of a non-trivial combination. □

This establishes that the prime coordinate system forms an authentic infinite-dimensional vector space over $\mathbb{Q}$ (or $\mathbb{R}$). Technically, we work in a countably-infinite-dimensional space where each integer has only finitely many nonzero coordinates, the space of sequences with finite support.

7.4 The True Vector Space

Under the logarithm, multiplication becomes addition:

$$\ln(n) = a_2 \cdot \ln(2) + a_3 \cdot \ln(3) + a_5 \cdot \ln(5) + \ldots$$

This is a genuine linear combination with:

- Basis vectors: {ln(2), ln(3), ln(5), ln(7), ...}
- Coordinates: (a_2, a_3, a_5, a_7, ...)

The linear independence proven above means this is not merely a notational convenience; it is exact mathematical structure.

7.5 Operations in Vector Space

Arithmetic	Vector Operation	Example
a × b	v(a) + v(b)	v(6) = v(2) + v(3)
a / b	v(a) − v(b)	v(6) − v(2) = v(3)
gcd(a, b)	min(v(a), v(b))	gcd(12, 18) = 6
lcm(a, b)	max(v(a), v(b))	lcm(12, 18) = 36
a^n	n · v(a)	v(8) = 3 · v(2)

Note: Division a/b corresponds to v(a) − v(b) only when b divides a (i.e., when all coordinates of v(b) are ≤ corresponding coordinates of v(a)). The positive integers under multiplication form a monoid, not a group; inverses do not generally exist within $\mathbb{Z}^+$.

PART IV: PROOF OF CONCEPT

8. The Coprimality Theorem

We demonstrate the complete framework by deriving a known number-theoretic result through all four representation layers.

Theorem 8.1. The probability that two randomly chosen positive integers are coprime equals $6/\pi^2 \approx 0.6079$.

Note: Since there is no uniform probability distribution on $\mathbb{Z}^+$, this theorem uses natural density: $P(\text{coprime}) = \lim_{N \to \infty} |\{(a,b) : 1 \leq a,b \leq N, \gcd(a,b) = 1\}| / N^2$. The limit exists and equals $6/\pi^2$.

8.1 Layer 1: English Formulation

Question: "What is the probability that two randomly chosen integers a and b share no common factors?"

Definition: Two integers are coprime if and only if gcd(a, b) = 1.

Equivalent statement: "For EVERY prime p, it is NOT the case that p divides BOTH a AND b."

This is a universal quantification over all primes, an infinite conjunction of conditions.

8.2 Layer 2: Probability Formulation

For a single prime p:

- P(p divides a random integer) = 1/p (every pth integer is divisible by p)
- P(p divides BOTH a AND b) = $1/p \times 1/p = 1/p^2$ (independence)
- P(p does NOT divide both) = $1 - 1/p^2$

For coprimality, ALL primes must fail to divide both. Divisibility by different primes is independent (by the Chinese Remainder Theorem: residue classes mod p and mod q are independent for distinct primes p, q). Therefore:

$$\mathrm{P}(\gcd(a,b) = 1) = \prod_{\{p \text{ prime}\}} (1 - 1/p^2)$$

8.3 Layer 3: Circuit Formulation

Each prime p defines a "filter" in the circuit:

- Conductance contribution: $G_p = 1 - 1/p^2$

- Resistance contribution: Ω_p = −ln(1 − 1/p²)

The filters are in SERIES (AND logic, all must pass):

$$\Omega\_\text{total} = \Sigma\_\{\text{p prime}\} -\ln(1 - 1/\text{p}^2)$$

Total conductance:

$$\text{G\_total} = \text{e}\hat{}(-\Omega\_\text{total}) = \prod\_\{\text{p prime}\} (1 - 1/\text{p}^2)$$

8.4 Layer 4: Vector Space Formulation

In prime coordinate space:

- Integer a has coordinates v_a = (a_2, a_3, a_5, ...) where a_p = exponent of p in a
- Integer b has coordinates v_b = (b_2, b_3, b_5, ...)

GCD operation = component-wise MINIMUM:

$$\text{v\_\{gcd\}} = \min(\text{v\_a}, \text{v\_b}) = (\min(a_2, b_2), \min(a_3, b_3), ...)$$

Coprime condition: v_{gcd} = (0, 0, 0, ...) = zero vector

For each dimension (prime p):

$$P(\min(\text{a\_p}, \text{b\_p}) = 0) = 1 - P(\text{both a\_p} \geq 1 \text{ and b\_p} \geq 1) = 1 - 1/\text{p}^2$$

8.5 Connection to the Zeta Function

The Euler product formula states:

$$\zeta(s) = \Sigma\_\{n=1\}\hat{}\{\infty\} \; 1/n\hat{}s = \prod\_\{\text{p prime}\} \; 1/(1 - \text{p}\hat{}(-s))$$

At s = 2:

$$\zeta(2) = \prod\_\{\text{p prime}\} \; 1/(1 - 1/\text{p}^2)$$

Therefore:

$$\prod\_\{\text{p prime}\} (1 - 1/\text{p}^2) = 1/\zeta(2)$$

And since ζ(2) = π²/6 (the Basel problem, proved by Euler in 1734):

$$P(\text{gcd(a,b)} = 1) = 1/\zeta(2) = 6/\pi^2 \approx 0.6079271...$$

8.6 Complete Translation Chain

Layer	Expression
English	"The probability that two random integers share no common factor"
Probability	P(coprime) = ∏_p (1 − 1/p²) = 1/ζ(2) = 6/π²
Circuit	G = 6/π² ≈ 0.608, Ω = ln(π²/6) ≈ 0.498
Vector	P(min(v_a, v_b) = $\vec{0}$), component-wise minimum is zero

8.7 Numerical Verification

Partial product (first 10 primes: 2, 3, 5, 7, 11, 13, 17, 19, 23, 29):

$$\prod_{p \le 29} (1 - 1/p^2) = (3/4)(8/9)(24/25)(48/49)... \approx 0.6123$$

As more primes are included, this converges to $6/\pi^2 \approx 0.6079$.

Monte Carlo verification: Sampling 1,000,000 random pairs (a, b) with a, b ∈ [1, 10000]:

Empirical P(coprime) ≈ 0.6080 ± 0.0005 ✓

8.8 Corollary: Computing π from the Lattice

Measuring the conductance of the infinite prime lattice yields π:

$$G = 6/\pi^2 \implies \pi = \sqrt{(6/G)}$$

The coprimality of integers, a purely combinatorial property, encodes the transcendental constant π through the structure of prime coordinate space.

PART V: DATABASE OPERATIONS

9. Relational Operations as Number Theory

The prime coordinate system provides a natural foundation for database operations. We demonstrate that fundamental relational algebra operations emerge directly from arithmetic.

9.1 The JOIN = GCD Theorem

Theorem 9.1. Set intersection (finding common elements between two sets) is equivalent to computing GCD.

We encode sets as squarefree integers, products where each prime appears at most once (exponent 0 or 1). Each element maps to a unique prime; presence in the set means the prime appears in the product.

Consider two sets encoded as products of their element-primes:

- Set A = {elements} → Product_A = ∏ prime(element)
- Set B = {elements} → Product_B = ∏ prime(element)
- A ∩ B = elements in both → GCD(Product_A, Product_B)

9.2 Worked Example: Finding Common Customers

English: "Which customers purchased both Product X and Product Y?"

Setup:

Table A (Bought X)	Table B (Bought Y)
Alice, Bob, Carol	Bob, Carol, Dave

Encoding (each customer assigned a unique prime):

- Alice = 2, Bob = 3, Carol = 5, Dave = 7

Tables as products:

- Table A = 2 × 3 × 5 = 30
- Table B = 3 × 5 × 7 = 105

Set intersection:

$$\mathrm{A} \cap \mathrm{B} = \mathrm{GCD}(30,\ 105) = 15 = 3 \times 5 = \{\mathrm{Bob},\ \mathrm{Carol}\}\ \checkmark$$

9.3 Four-Layer Translation

Layer	Expression
English	"Which customers bought both products?"
Set Theory	A ∩ B = {Bob, Carol}
Circuit	Common resistance paths = filters present in both circuits
Vector	min(v(30), v(105)) = min((1,1,1,0), (0,1,1,1)) = (0,1,1,0) → 15

9.4 Complete Relational Algebra

SQL Operation	Arithmetic	Vector Operation
Set Intersection / ∩	GCD(A, B)	min(v_A, v_B)
UNION / ∪	LCM(A, B)	max(v_A, v_B)
DIFFERENCE / −	A / GCD(A,B)	v_A − min(v_A, v_B)
CONTAINS?	B divides A?	v_A ≥ v_B per dim?

The Euclidean algorithm computes GCD in O(log n) operations, making intersection efficient. This unifies database theory with number theory: the structure of data operations emerges from the structure of integers.

Extension to multisets: The framework naturally extends to multisets (where elements can repeat) by allowing exponents greater than 1. The exponent of each prime encodes the count of that element. GCD then computes multiset intersection (minimum of counts), and LCM computes multiset union (maximum of counts).

Practical note: This encoding demonstrates the theoretical correspondence between set operations and arithmetic. A set of 100 elements would produce a ~200-digit number; practical implementations would require bounded prime assignments or alternative representations.

9.5 String Encoding Example

For completeness, we show how to encode arbitrary strings:

Encoding "CAT":

Position	Character	Value	Prime	Factor
1	C	3	2	$2^3 = 8$
2	A	1	3	$3^1 = 3$
3	T	20	5	5^{20}

Full encoding: encode("CAT") = $2^3 \times 3^1 \times 5^{20}$

Query: "What is the 2nd character?"

Extract exponent of prime(2) = 3: exponent = 1 → alphabet[1] = 'A' ✓

9.6 Universal Binary Encoding

The framework admits a universal encoding requiring no external lookup tables. Every binary string maps bijectively to a squarefree positive integer (one with no repeated prime factors).

Encoding rule: Position i (1-indexed from left to right, reading order) maps to the ith prime. Bit value becomes exponent (0 or 1). Note: this is reading order, not standard binary notation where the rightmost bit is least significant.

Binary	Interpretation	Product
"1"	2^1	2
"10"	$2^0 \times 3^1$	3
"11"	$2^1 \times 3^1$	6
"101"	$2^1 \times 3^0 \times 5^1$	10
"1101"	$2^1 \times 3^0 \times 5^1 \times 7^1$	70

Decoding: Factor the integer. Primes present → bit is 1. Primes absent → bit is 0.

This establishes binary as the natural language of the prime coordinate system. The encoding IS the mathematical structure; no external convention required.

Theorem 9.2 (Binary-Squarefree Bijection). The map from binary strings to squarefree integers is a bijection.

Proof. Each binary string of length n determines a unique subset of the first n primes (those positions with 1s). Each subset determines a unique squarefree product. Conversely, each squarefree integer factors uniquely into distinct primes, determining exactly which positions are 1. □

PART VI: COMPUTATIONAL COMPLETENESS

10. Turing Completeness

A computational system is Turing complete if it can simulate any Turing machine. We demonstrate that this framework satisfies all requirements through explicit construction.

10.1 Requirements

Requirement	Status	Implementation
Boolean completeness	✓ **SATISFIED**	AND, OR, NOT (Sections 3-5)
Conditional branching	✓ **SATISFIED**	Transistor gate (Section 10.3)
Unbounded memory	✓ **SATISFIED**	Infinite prime coordinates
Iteration	✓ **SATISFIED**	JUMP-IF-ZERO + GOTO (Section 10.3)

10.2 Counter Machine Simulation

Counter machines (Minsky machines) are known to be Turing complete. Our simulation encodes machine states as integers using Gödel numbering, the technique introduced by Gödel (1931) where prime factorization represents structured data. A two-counter machine has:

- Two counters C_1, C_2 holding non-negative integers
- Operations: INCREMENT(C_i), DECREMENT(C_i), JUMP-IF-ZERO(C_i, label), GOTO(label)

State Encoding:

We encode the machine state as an integer:

$$\text{State} = 2\hat{}(C_1) \times 3\hat{}(C_2) \times p\_k$$

where C_1 is the first counter, C_2 is the second counter, and p_k is a prime encoding the program counter (instruction number).

10.3 Operation Implementation

INCREMENT(C_1): Multiply state by 2

Circuit: Add Ω(2) = ln(2) in series

Vector: Add 1 to first coordinate: v → v + (1, 0, 0, ...)

DECREMENT(C_1): Divide state by 2 (if $C_1 > 0$)

Circuit: Subtract Ω(2) from total resistance

Vector: Subtract 1 from first coordinate: v → v − (1, 0, 0, ...)

JUMP-IF-ZERO(C_1, L): Branch based on whether $C_1 = 0$

This is the critical operation. We construct it explicitly:

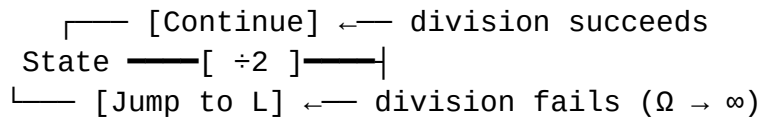
The Divisibility Test Circuit

The condition $C_1 = 0$ is equivalent to: 2 does NOT divide State.

Construction:

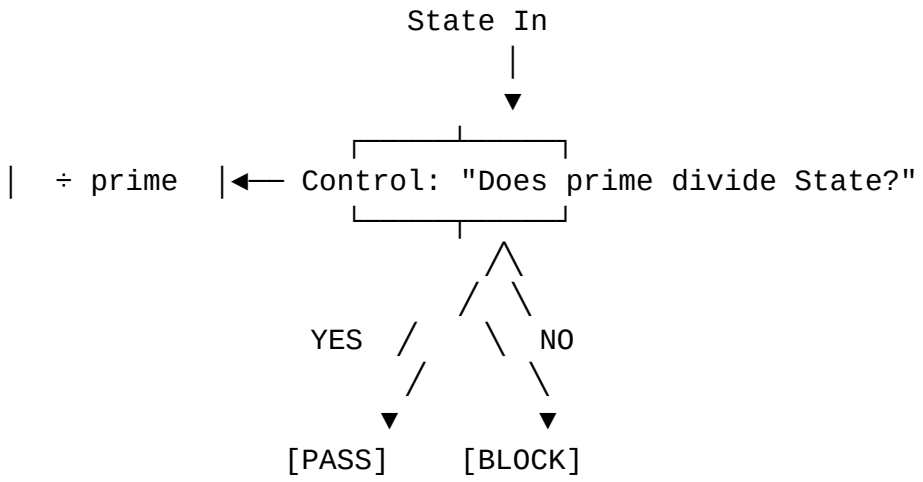
1. Attempt to compute State / 2
2. In vector space: v(State) − v(2) = v(State) − (1, 0, 0, ...)
3. If $v(State)_1 \geq 1$: subtraction succeeds → $C_1 > 0$ → continue
4. If $v(State)_1 = 0$: subtraction fails (would give negative) → $C_1 = 0$ → jump to L

Circuit implementation:



The division gate acts as a transistor: if the first coordinate is positive, the circuit conducts (finite resistance); if zero, the circuit blocks (infinite resistance), activating the alternative branch.

Transistor analogy (divisibility gate symbol):



Like a transistor with base/gate control, the divisibility gate has one input and two outputs: PASS (finite Ω, signal continues) or BLOCK (Ω → ∞, signal redirected).

Program Counter Manipulation

GOTO(L): Unconditional jump to instruction L

The program counter is encoded as a prime p_k where k is the instruction number. To jump from instruction k to instruction L:

$$\text{State\_new} = \text{State} \times (\text{p\_L} \ / \ \text{p\_k})$$

In resistance terms: subtract Ω(p_k) and add Ω(p_L). This replaces the current instruction prime with the target instruction prime while preserving the counter values.

10.4 Completeness Conclusion

Since counter machines are Turing complete and we can simulate them with explicit constructions for all operations (INCREMENT, DECREMENT, JUMP-IF-ZERO, and GOTO), the framework is Turing complete. The simulation uses Gödel numbering: machine state is encoded as $2^{C_1} \times 3^{C_2} \times p_k$, where prime exponents store counter values and a distinct prime encodes the program counter.

10.5 Computational Complexity Note

While Turing complete, the framework does not automatically solve NP-hard problems efficiently:

- Factorization bottleneck: For unstructured integers, decoding requires factoring. (Note: the counter machine simulation uses structured states of known form $2^a \times 3^b \times p_k$, where extracting exponents of known small primes is efficient.)
- NOT gate: Requires a reference signal (certainty P=1), analogous to a voltage reference in electronic inverters.
- Precision: For analog implementations using continuous probabilities, finite precision representation introduces limits not present in the discrete integer framework.

Undecidability: Since the framework is Turing complete, it inherits the halting problem. In circuit terms: determining whether a computation terminates is equivalent to asking whether the circuit resistance ever stabilizes to a halt state, and this question is undecidable in general.

PART VII: CONNECTIONS AND IMPLICATIONS

11. Primes as Irreducible Elements

Definition 11.1. An integer n > 1 is irreducible in the circuit sense if there do not exist integers a, b > 1 such that:

$$\Omega(\mathrm{n}) = \Omega(\mathrm{a}) + \Omega(\mathrm{b})$$

That is, the resistance ln(n) cannot be expressed as a sum of smaller resistances.

Theorem 11.2 (Prime Characterization). An integer is prime if and only if it is irreducible in the circuit sense.

Proof. n is reducible iff ∃ a, b > 1 with ln(n) = ln(a) + ln(b) = ln(ab) iff n = ab. This is precisely the definition of composite. Hence irreducible ≡ prime. □

This characterization shows that primality has a natural interpretation within the framework: primes are precisely those integers whose resistance cannot be decomposed into smaller components.

12. Connection to the Riemann Zeta Function

The Euler product formula provides a connection between our framework and the zeta function:

$$\zeta(\mathrm{s}) = \prod\_\{\mathrm{p\ prime}\}\ 1/(1 - \mathrm{p\widehat{}}(-\mathrm{s}))$$

Each prime p contributes a factor to the product. In our framework:

- Each factor 1/(1 − p^(−s)) corresponds to a geometric series of paths through the prime-p dimension
- At s = 2, the reciprocal yields the coprimality conductance $6/\pi^2$

The Euler product representation reflects the multiplicative structure of prime coordinates. The inverse function 1/ζ(s) = Σ μ(n)/n^s involves the Möbius function μ(n), which is zero for non-squarefree integers, directly connecting to our binary encoding where squarefree integers represent sets. Whether this connection can be extended to illuminate the distribution of zeta zeros remains an open question for future investigation.

13. Future Directions

1. Physical Implementation: Exploring whether circuits implementing these principles could offer computational advantages for probabilistic inference.
2. Zeta Zero Analysis: Investigating phase structure at zeta zeros through the lens of prime-composite interference patterns.

3. Möbius Function Connection: The inverse zeta function $1/\zeta(s) = \Sigma\ \mu(n)/n^s$, where μ is the Möbius function, appears closely related to our framework. The Möbius function equals zero for non-squarefree integers and ±1 for squarefree integers, mirroring our binary encoding. Investigating this connection may yield insights into multiplicative number theory.
4. AI Convergence Hypothesis: Testing whether trained embedding spaces converge toward prime coordinate structure as training data increases.
5. Higher-Dimensional Extensions: Investigating whether treating prime dimensions as spatial coordinates enables novel parallel computation models.

14. Conclusion

This paper has established a comprehensive framework for computation based on a rigorous isomorphism between probability mathematics and informational resistance. The key contributions include:

- Four-Layer Framework: English, probability, circuit, and vector space representations are demonstrated equivalent, enabling translation between intuitive descriptions and mathematical formulations.
- Complete Boolean Logic: AND (series), OR (parallel, both exclusive and inclusive), and NOT (phase-shift) form a functionally complete set, verified on binary truth tables.
- Coprimality Proof of Concept: Deriving $6/\pi^2$ through all four layers demonstrates the framework produces correct, non-trivial results connecting number theory to analysis.
- Database Operations: Relational JOIN, UNION, and INTERSECTION correspond to GCD, LCM, and related number-theoretic operations, revealing a unifying structure between data theory and arithmetic.
- Turing Completeness: Explicit construction of counter machine simulation, including the critical JUMP-IF-ZERO operation via divisibility testing.
- Prime Characterization: Primes correspond to the irreducible elements, configurations whose resistance cannot be decomposed into smaller components.

The framework demonstrates that computation need not be binary or discrete. Metaphorically, a system built on these principles acts as a "Physics Engine" for Logic: correct answers correspond to configurations of minimum informational resistance, much as physical systems settle into minimum-energy states. Whether this analogy can be made precise through actual physical implementation remains an open question.

References

[1] Shannon, C. E. (1948). A Mathematical Theory of Communication. Bell System Technical Journal, 27(3), 379-423.

[2] Riemann, B. (1859). Über die Anzahl der Primzahlen unter einer gegebenen Größe. Monatsberichte der Berliner Akademie.

[3] Euler, L. (1734). De summis serierum reciprocarum. Commentarii academiae scientiarum Petropolitanae 7: 123-134.

[4] Gödel, K. (1931). Über formal unentscheidbare Sätze der Principia Mathematica und verwandter Systeme I. Monatshefte für Mathematik und Physik, 38, 173-198.

[5] Minsky, M. L. (1967). Computation: Finite and Infinite Machines. Prentice-Hall.

[6] Cover, T. M., & Thomas, J. A. (2006). Elements of Information Theory (2nd ed.). Wiley-Interscience.

[7] Codd, E. F. (1970). A Relational Model of Data for Large Shared Data Banks. Communications of the ACM, 13(6), 377-387.

[8] Aczél, J. (1966). Lectures on Functional Equations and Their Applications. Academic Press.