FROM MATHEMATICAL PRINCIPLES OF LOGIC TO A CALCULUS OF UNCERTAINTY

Jay Kumar
Independent Researcher
Austin, TX
j.kumar.nbl@gmail.com

Keywords Mathematics · Logic · Uncertainty · Calculus · Beyond Binary

ABSTRACT

Classical formal logic provides a rule-governed foundation for truth, derivability, admissibility, and symbolic transformation. However, binary truth values are often insufficient for computational systems that must represent partial support, uncertainty, confidence, decay, and state transition. This paper develops a five-state calculus of uncertainty grounded in the formal lineage of Hilbert and Ackermann, Church, and Turing [2, 1, 5]. The proposed calculus represents logical states as tagged potential values over the interval $[0, 1]$, including false, almost false, undetermined, almost true, and true. Logical operators are defined over the underlying potential space and lifted back into named states through threshold-based classification. A minimal LambdaScript reference implementation is provided to demonstrate state construction, logical operators, smooth updates, decay, interpolation, and fuzzy membership functions [3]. The result is a compact computational substrate for uncertainty-aware admissibility and validation, intended as a stepping stone toward Beyond Binary systems and neurocontextual lateral propagation-diffusion architectures.

1 Introduction

Formal logic and computation share a common historical problem: how can reasoning be made exact, symbolic, and mechanically tractable? In the development of mathematical logic, this problem appears as the attempt to replace ambiguity in ordinary language with formal expressions, axioms, rules of transformation, and criteria for validity. In the development of computation, it appears as the attempt to characterize which procedures can be carried out by finite, effective means. These two lines of development converge through the work of Hilbert and Ackermann, Church, and Turing [2, 1, 5].

Hilbert and Ackermann's treatment of mathematical logic presents logic as a symbolic calculus. Sentences are represented by formal expressions, and compound statements are built through logical connectives such as negation, conjunction, disjunction, implication, and equivalence [2]. This makes truth-functional reasoning available to exact treatment. More importantly, it allows logical expressions to be transformed, normalized, and tested through formal rules. A formula may be reduced to normal form, inspected for validity or falsity, and studied through axiomatic systems. Once logic is treated in this way, reasoning becomes not merely a matter of interpretation, but a rule-governed symbolic process.

The axiomatic treatment of sentential calculus introduces a second level of formal structure. Instead of assuming ordinary inference, Hilbert and Ackermann specify axioms and rules by which further formulas may be derived [2]. This makes it possible to ask metatheoretic questions about the system itself: whether the axioms are consistent, whether they are independent, and whether the system is complete. Their use of nonstandard interpretations for independence proofs is especially important. In such proofs, logical symbols may be assigned custom value tables and designated values in order to show that one axiom cannot be derived from the others. This demonstrates that formal rigor does not require ordinary binary interpretation alone. What matters is that the value space, operations, and admissibility conditions are explicitly defined.

Church approaches the same general problem from the side of effective calculability. By identifying effectively calculable functions with recursive functions, and equivalently with lambda-definable functions, Church gives a formal account of what it means for a procedure to be mechanically calculable [1]. Lambda calculus then provides a symbolic system in which computation can be represented through abstraction, application, and reduction. In this setting, computation is not merely numerical calculation; it is formal transformation.

Turing gives another formal account of effective procedure through machines operating over symbols. A Turing machine has a finite configuration, a tape divided into squares, a scanned symbol, and a rule-governed transition from one complete configuration to the next [5]. Computation becomes a sequence of local symbolic transformations. Turing's universal machine extends this idea further by showing that machine descriptions themselves can become objects of computation. A machine may act not only on data, but on descriptions of procedures.

Together, these developments establish a foundation for computational logic. Hilbert and Ackermann show how formal reasoning can be represented through symbolic calculi, normal forms, axioms, and rules [2]. Church shows how effective procedure can be represented through recursive and lambda-definable functions [1]. Turing shows how computation can be represented as mechanical state transition over symbolic descriptions [5]. The present paper builds on this lineage by proposing a small uncertainty-state calculus as a computational extension of formal admissibility.

The motivation for this extension is that binary truth values are not always sufficient for systems that must represent partial support, uncertainty, unresolved state, confidence, decay, or graded admissibility. Classical sentential calculus evaluates formulas as true or false. However, many computational and intelligent systems must operate with states that are neither fully accepted nor fully rejected. Such systems require a way to represent intermediate states without abandoning formal constraint. This motivation is related to, but distinct from, the broader traditions of many-valued logic and fuzzy sets [4, 6].

This paper introduces a five-state uncertainty calculus consisting of false, almost false, undetermined, almost true, and true. Each state is represented as a tagged potential value over the interval $[0, 1]$. Logical operators are defined over the underlying potential values and then lifted back into named states through threshold-based classification. The resulting system is not intended to replace classical logic. Rather, it generalizes the idea of logical admissibility into a graded computational state space.

A minimal LambdaScript library is provided as a reference implementation. The library defines five named states, potential-based logical operators, smooth state updates, decay, interpolation, and optional fuzzy membership functions [3]. Its purpose is methodological rather than exhaustive: it demonstrates that uncertainty can be represented as a computable state surface governed by explicit operations.

This work is intended as a stepping stone toward two broader systems. The first is Beyond Binary, a framework for exploring computation beyond strictly binary state representation. The second is the Neurocontextual Lateral Propagation-Diffusion Framework, in which state, uncertainty, validation, and propagation are treated as interacting components of an adaptive architecture. In both cases, the aim is not to make computation vague, but to make uncertainty formal, computable, and available for validation.

The central claim of this paper is therefore simple: uncertainty need not weaken formal systems. When uncertainty is represented as state, governed by defined operators, and evaluated through admissibility conditions, it can extend formal logic into computational domains where binary truth alone is insufficient.

2 Background

The proposed uncertainty-state calculus is positioned at the intersection of formal logic, effective calculability, and machine computation. Its purpose is not to replace the classical foundations of logic and computation, but to extend one of their shared concerns: the formal treatment of admissible symbolic transformation. This section summarizes the three foundations most relevant to the present work: Hilbert and Ackermann's formal logic, Church's account of effective calculability, and Turing's machine model of computation [2, 1, 5].

2.1 Hilbert and Ackermann: Formal Logic, Normal Form, and Axiomatization

Hilbert and Ackermann present mathematical logic as a symbolic calculus. In this setting, ordinary sentences are replaced by formal expressions whose meanings are no longer dependent on the ambiguity of ordinary language [2]. Sentential calculus begins with elementary sentences and combines them through connectives such as negation, conjunction, disjunction, implication, and equivalence.

The truth value of a compound sentence is determined by the truth values of its components. This gives sentential calculus its truth-functional character. However, the importance of the calculus is not limited to assigning truth values. Logical expressions may also be transformed through equivalence-preserving rules. Implication can be replaced by disjunction and negation, biconditional expressions can be expanded into mutual implication, double negation can be eliminated, and De Morgan transformations can move negation inward [2].

Through these transformations, a formula may be reduced to normal form. Normal forms are important because they allow formulas to be inspected structurally. A formula can be analyzed to determine whether it is logically true, logically false, satisfiable, or unsatisfiable. In this way, sentential calculus becomes procedural: validity is approached through symbolic transformation followed by structural inspection.

Hilbert and Ackermann's axiomatic treatment adds a second layer. Certain logically true formulas are selected as axioms, and further formulas are generated through explicit rules such as substitution and implication [2]. This is significant because ordinary inference is no longer assumed informally. It is formalized as part of the calculus itself. The system can then be studied metatheoretically in terms of consistency, independence, and completeness.

Their independence proofs are especially relevant to the present work. To show that an axiom is independent, one may construct an interpretation in which the remaining axioms retain their designated value while the target axiom fails [2]. Such interpretations may use nonstandard value assignments and custom operation tables. This shows that formal rigor is not restricted to ordinary binary truth alone. A value system remains formal when its values, operations, and admissibility conditions are explicitly defined.

2.2 Church: Effective Calculability and Lambda-Definability

Church's work addresses the question of effective calculability. A function is effectively calculable when there is a definite procedure by which its values can be obtained. Church identifies this intuitive notion with recursive functions and with lambda-definable functions [1]. This identification provides a formal account of effective procedure.

Lambda calculus supplies a symbolic framework for this account. Computation is represented through abstraction, application, and reduction. Rather than treating computation only as arithmetic calculation, lambda calculus treats computation as formal expression transformation. A term is transformed according to rules, and the result of computation is reached through reduction [1].

This is relevant to the uncertainty-state calculus for two reasons. First, it shows that computation can be expressed as rule-governed symbolic transformation. Second, it provides a conceptual basis for implementing logical or mathematical calculi as executable systems. A calculus is not merely a notation; it can become an operational process.

The minimal reference implementation used in this paper follows this spirit. It represents uncertainty states and operators in LambdaScript, using symbolic definitions to define state construction, logical operations, smooth updates, decay, interpolation, and membership behavior [3]. The goal is not to reproduce Church's lambda calculus directly, but to use the same general idea: formal definitions can be made computational.

2.3 Turing: Computation as Machine State Transition

Turing approaches effective procedure through a different formal model. A computing machine is supplied with a tape divided into squares. At any stage, the machine has a configuration, scans a symbol, and acts according to a finite table of rules [5]. Its action may include writing, changing configuration, and moving to a neighboring square. Computation is therefore represented as a sequence of complete configurations.

This account makes computation local, mechanical, and state-based. The machine does not operate by intuition or global understanding. It acts by reading a symbol, applying a rule, and transitioning to a new configuration. Turing also distinguishes between output symbols and intermediate symbols. Output symbols form the produced sequence, while intermediate symbols function as scratch work used during computation [5].

Turing's use of skeleton tables and machine-configuration functions also introduces an important abstraction mechanism. Repeated machine processes can be abbreviated and later expanded into complete tables [5]. This shows that abstraction can exist inside a formal machine description without adding new computational power. It compresses repeated structure while preserving reducibility to primitive machine steps.

The universal machine extends this principle further. A machine description may itself become data for another machine [5]. This means that computation can operate not only over symbols, but over descriptions of symbolic procedures. In modern terms, this anticipates interpreters, virtual machines, compilers, and programmable architectures.

For the present work, Turing's model contributes the idea of computation as explicit state transition. An uncertainty-state calculus must similarly define how states are represented and how operations transform one state into another. The five-state system developed here treats uncertainty as part of computational state rather than as an external annotation.

2.4 From Binary Truth to Admissible State

The classical systems above share a common feature: they make symbolic transformation exact. Hilbert and Ackermann formalize logical expressions, rules, normal forms, and axioms [2]. Church formalizes effective calculation through recursion and lambda-definability [1]. Turing formalizes computation as machine transition over symbols and descriptions [5].

The present paper extends this shared pattern into uncertainty-state computation. Classical sentential calculus begins with binary truth values. A formula is true or false under a given interpretation, and its validity can be checked through truth tables, normal forms, or axiomatic derivation. However, many computational systems require intermediate or unresolved states. A state may be partially supported, uncertain, unstable, decaying, context-dependent, or admissible only above a threshold.

The proposed five-state calculus introduces a small formal value space:

[false, almost false, undetermined, almost true, true.]

Each state is represented by a potential value in the interval:

[[0,1].]

The purpose of this representation is not to replace truth with vagueness. The purpose is to make uncertainty explicit and computable. Logical operators are defined over potential values, and the resulting values are mapped back into named states through thresholds. This preserves the essential formal requirement: the value space, operations, and admissibility conditions are explicitly defined.

2.5 Relation to Existing Logical Extensions

The five-state calculus has obvious surface similarities to many-valued logic and fuzzy logic [4, 6]. However, the focus of this paper is narrower and more computationally oriented. The system is not introduced as a full alternative logical foundation. It is introduced as a minimal uncertainty-state substrate for computation.

Many-valued logic shows that logical systems need not be restricted to two truth values [4]. Fuzzy logic shows that graded membership can be useful for reasoning under partial truth or imprecision [6]. The present work uses a related intuition, but frames the problem through admissibility, state transition, and computational validation. The five states are not merely truth labels. They are named regions over a potential surface, and operators act as state transformations.

This distinction is important for later applications. In Beyond Binary systems, the concern is not only whether a proposition is true, but whether a computational state can be represented beyond binary opposition. In neurocontextual lateral propagation-diffusion architectures, the concern is not only whether a signal is accepted or rejected, but how it propagates, decays, strengthens, weakens, or remains unresolved under validation.

2.6 Methodological Position

The methodology of this paper follows from the preceding foundations. From Hilbert and Ackermann, it adopts the requirement that operations and admissibility conditions be explicit [2]. From Church, it adopts the idea that symbolic calculi can be operationalized as computation [1]. From Turing, it adopts the view that computation proceeds through defined state transitions [5].

The result is a minimal five-state uncertainty calculus implemented as a small LambdaScript library [3]. The library is not intended to model full intelligence, replace classical logic, or solve the general problem of reasoning under uncertainty. Its purpose is to demonstrate a compact computational substrate in which uncertainty is represented as state and transformed by explicit operators.

This provides a stepping stone toward larger systems, including Beyond Binary and neurocontextual lateral propagation-diffusion. In those systems, uncertainty is not treated as noise to be removed. It is treated as part of the state surface to be represented, transformed, and validated.

3 A Five-State Calculus of Uncertainty

3.1 Motivation

Classical sentential logic assigns formulas truth values. In its simplest form, a proposition is evaluated as either true or false. Many-valued and fuzzy systems extend this view by allowing intermediate values or degrees of membership. The present calculus adopts a different emphasis. It treats the value of a proposition not only as a point, but as a state trajectory.

The motivation is that uncertainty in computational systems is often dynamic. A state may begin as undetermined, move toward support, weaken, reverse, decay, or converge. A static truth value does not capture this motion. A value close to truth may be unstable, while a value near truth with very low residual velocity may represent a converged state. Similarly, a value near the center may be unresolved, balanced between opposing evidence, or rapidly moving toward one pole.

The proposed calculus therefore represents each proposition as a signed state function over time:

$$s(t) \in [-1, 1].$$

The poles of the interval represent resolved states:

[s(t) = -1 false,]

[s(t) = 0 undetermined,]

[s(t) = 1 true.]

Under this interpretation, truth is not merely a discrete Boolean assignment. It is the limiting behavior of a state trajectory. A proposition does not simply become true or false because it crosses a fixed point. Rather, it approaches, stabilizes, or collapses toward a pole under the influence of evidence, decay, contradiction, or validation.

3.2 State Space

The calculus defines five named states:

$$\mathcal{S} = \{\text{false}, \text{almost false}, \text{undetermined}, \text{almost true}, \text{true}\}.$$

These names are labels over a signed state axis:

$$s(t) \in [-1, 1].$$

The value (s(t)) represents the current signed position of a proposition. Negative values indicate motion or tendency toward falsity. Positive values indicate motion or tendency toward truth. Values near zero represent unresolved or undetermined state.

The five-state interpretation may be written schematically as:

State region	Name
$s(t) \ll 0$	false
$s(t) < 0$	almost false
$s(t) \approx 0$	undetermined
$s(t) > 0$	almost true
$s(t) \gg 0$	true

This table is intentionally schematic. The exact boundaries are not fixed by the calculus itself. They may be selected according to the needs of a particular implementation, domain, or validation system. The central requirement is that the state space, transition rules, and admissibility criteria be explicitly defined.

3.3 Signed Potential and Normalized Potential

The signed state value (s(t)) is the primary theoretical representation. However, an equivalent normalized potential (p(t)) over ([0,1]) may also be used:

$$p(t) = \frac{s(t) + 1}{2}.$$

The inverse mapping is:

[s(t) = 2p(t)-1.]

Thus:

$$s(t) = -1 \iff p(t) = 0,$$

$$s(t) = 0 \iff p(t) = 0.5,$$

$$s(t) = 1 \iff p(t) = 1.$$

The normalized representation is useful for implementations that already operate over ([0,1]), while the signed representation is conceptually clearer for uncertainty because it gives the undetermined state a true center.

3.4 Uncertainty, Certainty, and Polarity

The signed state value separates three concepts: polarity, certainty, and uncertainty.

Polarity is the direction of the state:

$$\text{polarity}(s) = \text{sign}(s).$$

A negative polarity indicates tendency toward false. A positive polarity indicates tendency toward true. Zero indicates no resolved polarity.

Certainty is the distance from the undetermined center:

[c(s) = |s|.]

Uncertainty is maximal at the center and minimal at the poles:

[u(s) = 1 - |s|.]

Thus:

[u(0)=1,]

[u(-1)=u(1)=0.]

This captures the core intuition of the system. The center is not half true. It is maximally unresolved. The poles are not merely extreme values. They are states of maximal resolution.

3.5 State Velocity

A static state value is not sufficient to determine resolution. The system must also account for motion. The velocity of a state is defined as:

$$v(t) = \frac{ds}{dt}.$$

If:

[v(t) > 0,]

the state is moving toward truth. If:

[v(t) < 0,]

the state is moving toward falsity. If:

$$v(t) \approx 0,$$

the state may be stable, stalled, balanced, or converged.

This distinction matters because position alone does not determine the meaning of a state. A value near (1) with near-zero velocity may represent a state that has already converged toward truth. A value near (0) with near-zero velocity may represent unresolved balance. A value near (0) with high positive velocity may represent active motion toward truth. Therefore, a state must be interpreted by both its position and its motion.

3.6 Acceleration and Resolution Pressure

The acceleration of a state is defined as:

$$a(t) = \frac{d^2 s}{dt^2}.$$

Acceleration measures change in velocity. If velocity represents movement toward a pole, acceleration represents change in the strength or direction of that movement. A positive acceleration may indicate increasing support for truth. A negative acceleration may indicate increasing support for falsity, weakening support for truth, or reversal of prior motion.

The calculus does not require every implementation to use acceleration. However, acceleration becomes useful when distinguishing between stable convergence, rapid collapse, oscillation, and reversal.

3.7 Distance Traveled and Accumulated Motion

Resolution is also affected by path history. A state that briefly touches a region near truth should not necessarily be treated the same as a state that has persistently moved toward truth over time. To capture this, the calculus defines accumulated motion.

The total distance traveled over an interval $[t_0, t]$ is:

$$D(t) = \int_{t_0}^{t} |v(\tau)|, d\tau.$$

This measures the total amount of state movement, regardless of direction.

Directional motion can be separated into trueward and falseward components:

$$v^+(t) = \max(v(t), 0),$$

$$v^-(t) = \max(-v(t), 0).$$

The accumulated trueward motion is:

$$T(t) = \int_{t_0}^{t} v^+(\tau), d\tau.$$

The accumulated falseward motion is:

$$F(t) = \int_{t_0}^{t} v^-(\tau), d\tau.$$

These quantities allow the system to distinguish between momentary position and sustained directional movement. A state may be classified not only by where it is, but by how it arrived there.

3.8 Resolution and Collapse

A state is resolved when its trajectory approaches a pole with sufficient stability, motion, or accumulated evidence. Resolution may occur in two forms.

The first is active resolution. A state is actively resolving when it is moving toward a pole with significant velocity:

[s(t) > 0, v(t) > 0]

for motion toward truth, or:

[s(t) < 0, v(t) < 0]

for motion toward falsity.

The second is settled resolution. A state is settled when it is near a pole and has low residual velocity:

$$s(t) \approx 1, \quad v(t) \approx 0$$

for settled truth, or:

$$s(t) \approx -1, \quad v(t) \approx 0$$

for settled falsity.

Thus, a low velocity does not always indicate weak resolution. Near the center, low velocity may indicate unresolved balance. Near a pole, low velocity may indicate convergence.

A general resolution score may be defined as:

$$\rho(t) = |s(t)|, |v(t)|.$$

This quantity is high when the state is both far from the undetermined center and moving strongly. For settled states, a separate convergence criterion is needed:

$$|s(t)| \to 1 \quad \text{and} \quad |v(t)| \to 0.$$

The calculus therefore treats resolution as dynamic rather than purely positional. Collapse is not only crossing a threshold. Collapse is convergence of a trajectory toward a stable pole.

3.9 Five-State Classification

The five named states can now be interpreted dynamically.

Name	Dynamic interpretation
false	trajectory resolved or converging toward -1
almost false	trajectory leaning or moving toward falsity
undetermined	trajectory near 0 or unresolved by motion
almost true	trajectory leaning or moving toward truth
true	trajectory resolved or converging toward 1

This classification can be implemented through thresholds, but thresholds are not the essence of the calculus. The essence is that every expression produces a state trajectory. The named states are interpretable labels assigned to the behavior of that trajectory.

3.10 Logical Operators

Logical operators may be defined over signed state values. A simple negation operator reverses polarity:

$$\mathrm{NOT}(s) = -s.$$

Conjunction and disjunction may be defined in several ways depending on whether the implementation emphasizes minimum support, maximum support, product interaction, or another admissibility rule. A minimal signed version may use normalized values.

Given:

$$p = \frac{s+1}{2},$$

classical fuzzy-style operators may be applied over (p):

$$\mathrm{AND}_p(p_a, p_b) = \min(p_a, p_b),$$

$$\mathrm{OR}_p(p_a, p_b) = \max(p_a, p_b),$$

$$\mathrm{IMPLY}_p(p_a, p_b) = \max(1 - p_a, p_b).$$

The resulting normalized value may then be mapped back into signed state:

[s = 2p-1.]

This gives a practical bridge between signed uncertainty trajectories and normalized operator implementations.

However, the dynamic interpretation adds another layer. An operator may combine not only positions, but also velocities. For propositions (A) and (B), an expression may have:

$$s_A(t), \quad v_A(t)$$

and:

$$s_B(t), \quad v_B(t).$$

The resulting expression has its own trajectory:

$$s_{A \circ B}(t),$$

where $\circ$ is a logical operator. The state of a compound expression is therefore not merely a static value. It is a graph produced by combining the trajectories of its components.

3.11 State Update Operators

A discrete update rule may be written as:

$$s_{t+1} = \mathrm{clamp}(s_t + \Delta_t, -1, 1),$$

where Δ_t represents new evidence, validation, contradiction, or decay pressure.

A continuous version may be written as:

$$\frac{ds}{dt} = E_T(t) - E_F(t) - \lambda s(t),$$

where $E_T(t)$ represents trueward evidence, $E_F(t)$ represents falseward evidence, and $\lambda s(t)$ represents decay toward the undetermined center. This decay term is important because loss of support should not automatically imply falsity. Unsupported states drift toward uncertainty, not toward falsehood. A target-attraction version may also be written as:

$$\frac{ds}{dt} = k(q - s),$$

where $q \in [-1, 1]$ is the target state and (k) is a convergence rate. This produces smooth motion toward a pole or intermediate target. It models state resolution as gradual convergence rather than immediate Boolean collapse.

3.12 Membership Functions

The five named states may also be represented through membership functions over the signed interval ([-1,1]). These functions are useful when a state has partial membership in neighboring regions.

For example, triangular or ramp functions may be assigned to false, almost false, undetermined, almost true, and true. This allows a state near a boundary to partially belong to adjacent categories. However, in the present calculus, membership is secondary to trajectory. Membership describes where the state is. Velocity and accumulated motion describe how the state is resolving.

4 Reference Implementation

4.1 LambdaScript Library

A minimal reference implementation is provided in LambdaScript. The purpose of the implementation is to demonstrate that the proposed uncertainty calculus can be represented as executable symbolic definitions rather than as informal terminology.

The implementation defines named uncertainty states, state construction, state extraction, potential-based logical operators, state updates, decay, interpolation, and optional membership functions. It is intentionally small. Its goal is not to provide a complete reasoning engine, but to show that uncertainty states can be represented and transformed by explicit computational rules.

The initial implementation may operate over normalized potentials:

$$p \in [0, 1].$$

This representation is equivalent to the signed state model through the mappings:

$$p = \frac{s+1}{2},$$

$$s = 2p - 1.$$

Thus, an implementation over ([0,1]) can still support the signed theoretical interpretation. In future versions, the library may use signed state values directly.

4.2 State Representation

Each state is represented as a tagged pair:

$$\mathrm{PAIR}(\mathrm{name}, \mathrm{potential}).$$

The named states are:

Name	p	s
false	0	-1
almost false	0.25	-0.5
undetermined	0.5	0
almost true	0.75	0.5
true	1	1

In LambdaScript-style notation, this may be represented as:

```
FALSE5 = PAIR FALSE5_STATE 0
ALMOST_FALSE5 = PAIR ALMOST_FALSE5_STATE 0.25
UNDETERMINED5 = PAIR UNDETERMINED5_STATE 0.5
ALMOST_TRUE5 = PAIR ALMOST_TRUE5_STATE 0.75
TRUE5 = PAIR TRUE5_STATE 1
```

The functions:

```
STATE_NAME
```

and:

`STATE_POT`

extract the symbolic state name and numeric potential.

4.3 Threshold-Based State Assignment

The implementation assigns named states from a numeric potential by clamping the value into $[0, 1]$ and applying thresholds. In the initial implementation, these thresholds are:

$$0.125, \quad 0.375, \quad 0.625, \quad 0.875.$$

These values divide the normalized interval into five regions centered around the anchor values:

$$0, \quad 0.25, \quad 0.5, \quad 0.75, \quad 1.$$

This gives a simple classifier from potential value to named state. In the theoretical model, these thresholds are implementation parameters rather than fixed logical constants. They may be adjusted according to domain, tolerance, validation requirements, or convergence behavior.

4.4 Operator Implementation

The reference implementation defines operators over numeric potential values and then lifts the result back into a named state.

Negation is defined by complement:

$$\mathrm{NOT}_p(p) = 1 - p.$$

Conjunction is defined by minimum:

$$\mathrm{AND}_p(a, b) = \min(a, b).$$

Disjunction is defined by maximum:

$$\mathrm{OR}_p(a, b) = \max(a, b).$$

Exclusive difference may be defined by absolute difference:

$$\mathrm{XOR}_p(a, b) = |a - b|.$$

Implication may be defined as:

$$\mathrm{IMPLY}_p(a, b) = \max(1 - a, b).$$

Equivalence may be defined as:

$$\mathrm{EQUIV}_p(a, b) = 1 - |a - b|.$$

The lifted operators have the form:

$$\mathrm{OP}_5(x, y) = \mathrm{STATE\_FROM\_POT}\left(\mathrm{OP}_p\left(\mathrm{STATE\_POT}(x), \mathrm{STATE\_POT}(y)\right)\right).$$

This separates numeric operation from state labeling. Operators act on the underlying potential, and the resulting value is classified back into the five-state system.

4.5 Dynamic State Updates

The reference implementation includes update operations that support the dynamic interpretation of uncertainty.

A shift operation adjusts a state by a numeric amount:

$$p' = \mathrm{clamp}(p + \Delta, 0, 1).$$

A decay operation reduces a value over time:

$$p' = pe^{-\lambda t}.$$

An interpolation operation moves between two states:

$$p' = (1 - \alpha)p_a + \alpha p_b.$$

These operations are implementation-level approximations of the dynamic theory. In the signed model, analogous operations apply to $s(t)$ over $[-1, 1]$. For example, a signed shift is:

$$s' = \mathrm{clamp}(s + \Delta, -1, 1).$$

A decay-to-uncertainty operation may be written as:

$$s' = se^{-\lambda t}.$$

This causes unsupported signed states to drift toward 0, the undetermined center.

4.6 Examples

The implementation supports examples such as:

$$\mathrm{STATE\_FROM\_POT}(0.98) = \text{true},$$

$$\mathrm{STATE\_FROM\_POT}(0.70) = \text{almost true},$$

$$\mathrm{STATE\_FROM\_POT}(0.50) = \text{undetermined},$$

$$\mathrm{STATE\_FROM\_POT}(0.20) = \text{almost false},$$

$$\mathrm{STATE\_FROM\_POT}(0.01) = \text{false}.$$

It also supports compound operations such as:

$$\mathrm{AND}_5(\text{almost true}, \text{undetermined}),$$

$$\mathrm{OR}_5(\text{almost false}, \text{undetermined}),$$

$$\mathrm{NOT}_5(\text{almost true}),$$

and state movement such as:

$$\mathrm{SHIFT}_5(\text{undetermined}, 0.2).$$

These examples demonstrate that the library can construct, transform, and classify uncertainty states.

4.7 Reproducibility

The reference implementation is small enough to be inspected directly. Its definitions are intended to be reproducible, modifiable, and portable. The normalized implementation over $[0, 1]$ provides a practical executable substrate, while the signed interpretation over $[-1, 1]$ provides the theoretical model used in this paper.

Future versions may extend the implementation with explicit signed-state support, velocity tracking, accumulated directional motion, convergence detection, and resolution scoring. In such an implementation, each proposition would be represented not only by a current state value, but by a state trajectory:

$$s_p(t),$$

with derivative:

$$v_p(t) = \frac{ds_p}{dt}.$$

This would allow the implementation to distinguish between unresolved states, actively resolving states, and settled states.

5 Computational Interpretation

5.1 Admissibility as State Validation

In classical formal logic, a formula may be evaluated for truth, validity, satisfiability, or derivability. In the present calculus, the corresponding computational question is whether a state is admissible under a given interpretation, context, or validation condition.

Admissibility is not identical to truth. A state may be admissible even when it is not fully true or false. For example, an undetermined state may be admissible when the system lacks sufficient evidence to collapse toward either pole. Similarly, an almost true state may be admissible when it has accumulated support but has not yet converged to full resolution.

The proposed calculus therefore separates truth from admissibility. Truth and falsity are poles of the signed state space, while admissibility is a validation condition over the state trajectory. A state may be accepted, rejected, delayed, decayed, or propagated depending on its position, velocity, accumulated motion, and context.

Let:

$$s(t) \in [-1, 1]$$

represent the signed state of a proposition or expression. A validation function may be written as:

$$V(s, t, C) \rightarrow \text{accept}, \text{reject}, \text{defer}, \text{propagate},$$

where (C) represents context. This allows the system to distinguish between a state that is false, a state that is unresolved, and a state that is not yet admissible for propagation.

In this sense, admissibility is a computational analogue of logical acceptability. It does not merely ask whether a proposition is true. It asks whether the current state of the proposition is valid enough, stable enough, or convergent enough to be used by the system.

5.2 Uncertainty as Computable Potential

The calculus treats uncertainty as a computable quantity rather than as an informal description. In the signed model, uncertainty is highest at the center of the state interval and lowest at the poles.

The uncertainty of a state is defined as:

$$u(s) = 1 - |s|.$$

The certainty of a state is defined as:

$$c(s) = |s|.$$

Thus, when:

$$s = 0,$$

the state is maximally undetermined:

$$u(0) = 1, \quad c(0) = 0.$$

When:

$$s = -1 \quad \text{or} \quad s = 1,$$

the state is maximally resolved:

$$u(s) = 0, \quad c(s) = 1.$$

This representation avoids treating uncertainty as a third static truth value. Instead, uncertainty becomes a function of distance from resolution. The state may move, decay, accelerate, oscillate, or converge. The value of uncertainty changes as the trajectory changes.

This makes uncertainty computational. It can be stored, updated, compared, thresholded, decayed, and propagated. A system can therefore reason not only over propositions, but over the changing uncertainty surfaces associated with those propositions.

5.3 Discrete States and Continuous Potentiation

The five named states provide a discrete vocabulary:

$$\text{false}, \text{almost false}, \text{undetermined}, \text{almost true}, \text{true}.$$

However, the underlying state value is continuous:

$$s(t) \in [-1, 1].$$

This allows the calculus to combine discrete interpretability with continuous motion. The named states are useful for symbolic reasoning, reporting, and implementation. The continuous state value is useful for modeling gradual movement, decay, convergence, and transition.

The system may therefore be interpreted at two levels.

At the symbolic level, a proposition may be labeled:

$$\text{almost true}$$

or:

$$\text{undetermined}.$$

At the dynamic level, the same proposition has a trajectory:

$$s_p(t),$$

with velocity:

$$v_p(t) = \frac{ds_p}{dt}.$$

This distinction is important. The label is not the full computational state. It is a readable classification of a point or region on a trajectory. The full computational state includes position, motion, and history.

Thus, the calculus avoids forcing a choice between discrete and continuous representation. It uses discrete labels for interpretability and continuous potentiation for computation.

5.4 State Resolution as Convergence

In a Boolean system, evaluation collapses directly into one of two values. In the proposed calculus, resolution is modeled as convergence toward a pole.

A state may actively resolve when it moves toward a pole with sufficient velocity:

$$s(t) > 0, \quad v(t) > 0$$

for trueward motion, or:

$$s(t) < 0, \quad v(t) < 0$$

for falseward motion.

A state may also be settled when it has approached a pole and its residual motion has become small:

$$|s(t)| \to 1,$$

$$|v(t)| \to 0.$$

This distinction matters because low velocity has different meanings depending on position. Near the center, low velocity may indicate unresolved balance or lack of evidence. Near a pole, low velocity may indicate convergence.

State resolution is therefore not defined by position alone. It depends on the relationship among position, velocity, accumulated motion, and context. A value near a pole with low residual motion may represent a collapsed state. A value near the center with low residual motion may represent an unresolved state.

This gives the calculus a dynamic interpretation:

$$\text{truth} \approx \text{convergence toward } +1,$$

$$\text{falsity} \approx \text{convergence toward } -1,$$

$$\text{undetermined} \approx \text{persistence near } 0 \text{ or unresolved motion.}$$

5.5 Trajectory-Based Expression Evaluation

In the proposed calculus, every expression may be interpreted as producing a trajectory rather than a single Boolean value. If (A) is a proposition, then its state is:

$$s_A(t).$$

If (B) is another proposition, then its state is:

$$s_B(t).$$

For a compound expression:

$$A \circ B,$$

where $\circ$ is a logical operator, the resulting expression has its own state trajectory:

$$s_{A \circ B}(t).$$

The operator determines how the component trajectories combine. A static implementation may combine only current positions:

$$s_A(t), \quad s_B(t).$$

A dynamic implementation may also combine velocities:

$$v_A(t), \quad v_B(t),$$

and accumulated motion:

$$T_A(t), \quad F_A(t), \quad T_B(t), \quad F_B(t).$$

This means that expression evaluation can be extended from Boolean truth-value assignment to trajectory transformation. A logical expression becomes a graph of state over time. The interpretation of the expression depends not only on where the graph is, but on how it moves and whether it converges.

5.6 Relation to Many-Valued Logical Matrices

Many-valued logic extends classical logic by allowing formulas to take values beyond true and false. Logical matrices define a set of values, designated values, and operations over those values. This provides an important precedent for non-binary logical systems.

The present calculus shares the idea that logical systems may use more than two values. However, it differs in emphasis. The five named states are not merely elements of a static truth table. They are labels over a dynamic signed state space.

In a many-valued matrix, an expression evaluates to a value. In the present calculus, an expression evaluates to a trajectory:

$$s(t).$$

The named value at any moment is a classification of that trajectory. This allows the system to represent unresolved motion, active resolution, settled convergence, reversal, decay, and oscillation.

Thus, the calculus may be viewed as a trajectory-based extension of many-valued interpretation. It inherits the formal lesson that values and operations must be explicitly defined, but it adds temporal movement as part of the computational object.

5.7 Operational Meaning

Operationally, the calculus may be interpreted as a state machine over signed potentials. Each proposition or expression has a current state value, and each update changes that value according to evidence, contradiction, decay, or validation.

A minimal operational state may contain:

$$(s, v, c, u),$$

where:

$$s = \text{signed state},$$

$$v = \text{state velocity},$$

$$c = \text{certainty},$$

$$u = \text{uncertainty}.$$

A richer implementation may also store:

$$T(t), \quad F(t), \quad D(t),$$

where (T(t)) is accumulated trueward motion, (F(t)) is accumulated falseward motion, and (D(t)) is total distance traveled.

Under this interpretation, uncertainty-aware computation becomes stateful. The system is not only evaluating propositions. It is tracking how propositions move through state space and whether their motion is admissible for further propagation.

5.8 Why This Is Computational

The proposed calculus is computational because each part of the model can be represented and updated by explicit operations.

The signed state value can be stored as a number:

$$s \in [-1, 1].$$

Velocity can be approximated by finite difference:

$$v_t = \frac{s_t - s_{t-1}}{\Delta t}.$$

Acceleration can be approximated by:

$$a_t = \frac{v_t - v_{t-1}}{\Delta t}.$$

Uncertainty and certainty can be computed directly:

$$u(s) = 1 - |s|,$$

$$c(s) = |s|.$$

Accumulated motion can be approximated by discrete sums:

$$D_t = \sum_i |v_i| \Delta t,$$

$$T_t = \sum_i \max(v_i, 0) \Delta t,$$

$$F_t = \sum_i \max(-v_i, 0) \Delta t.$$

These definitions allow the system to be implemented in ordinary computational terms. No appeal to informal intuition is required once the value space, operators, update rules, and admissibility conditions are specified.

5.9 Interpretive Summary

The computational interpretation of the calculus may be summarized as follows:

Boolean logic: proposition $\rightarrow$ truth value.

Many-valued logic: proposition $\rightarrow$ value from a finite set.

Five-state uncertainty calculus: proposition $\rightarrow$ signed state trajectory.

The five-state labels provide readable classifications. The signed state value provides continuous position. The derivative provides velocity of resolution. Accumulated motion provides path history. Validation determines whether a state is admissible for use, propagation, or collapse.

The result is a computational model in which uncertainty is not a failure of evaluation. It is part of the evaluated state.

6 Applications

6.1 Beyond Binary

6.2 Neurocontextual Lateral Propagation-Diffusion

6.3 Validation and Propagation in Stateful Architectures

7 Methodology

The methodology of this paper consists of three stages: formal abstraction, computational representation, and implementation-level observation.

First, the paper abstracts from classical sentential logic and computability theory to define uncertainty as a formal state rather than as an informal descriptor. The abstraction begins with the distinction between binary truth values and unresolved computational states. A proposition is represented not only by a value, but by a signed state trajectory:

$$s(t) \in [-1, 1].$$

The value (-1) represents resolved falsity, (0) represents an undetermined state, and (1) represents resolved truth. This signed model allows the calculus to distinguish between polarity, certainty, uncertainty, and motion.

Second, the paper defines computable quantities over this state space. Certainty is represented by distance from the undetermined center:

$$c(s) = |s|.$$

Uncertainty is represented by remaining distance from resolution:

$$u(s) = 1 - |s|.$$

State velocity is represented by:

$$v(t) = \frac{ds}{dt}.$$

This permits the system to distinguish between a state that is unresolved, a state that is actively resolving, and a state that has converged near a pole.

Third, a minimal LambdaScript reference implementation is used to test whether the proposed state structure can be expressed as executable definitions. The implementation defines five named states, potential extraction, state construction, logical operators, state shifting, decay, interpolation, and membership functions. The implementation currently operates over normalized potentials $p \in [0, 1]$, with the signed interpretation recovered through:

$$s = 2p - 1.$$

The purpose of the implementation is not to demonstrate full-scale reasoning or empirical performance. Rather, it functions as a reference experiment: it tests whether the theoretical state model can be reduced to compact symbolic definitions and whether those definitions produce interpretable state transformations.

7.1 Experimental Procedure

The experimental procedure consists of the following steps.

First, the five named states are defined as tagged values with associated numeric potentials. Second, threshold-based classification is used to map numeric potentials into state labels. Third, logical operators are applied to the numeric potentials and lifted back into named states. Fourth, state update operators are applied to observe movement between states. Fifth, decay and interpolation are used to test whether states can move gradually rather than collapse immediately into Boolean values.

The reference implementation is evaluated qualitatively by inspecting whether the resulting states remain interpretable under the proposed calculus. The central question is not whether the implementation outperforms another system, but whether it demonstrates a coherent computational substrate for uncertainty-state representation.

8 Observations

Several observations follow from the reference implementation and formal model.

First, the five-state vocabulary provides a readable surface over a continuous state space. The labels false, almost false, undetermined, almost true, and true are useful for interpretation, but they do not exhaust the computational state. The underlying potential value remains available for finer-grained computation.

Second, the signed model clarifies the meaning of uncertainty. In the normalized model $p \in [0, 1]$, the value (0.5) may appear to represent partial truth. In the signed model $s \in [-1, 1]$, the corresponding value (s=0) is more clearly interpreted as undetermined. This avoids treating uncertainty as half truth. Instead, uncertainty is represented as lack of resolved polarity.

Third, state motion is essential. A static value does not always distinguish between unresolved, resolving, and settled states. A state near a pole with low velocity may represent convergence, while a state near the center with low velocity may represent unresolved balance. This suggests that uncertainty-aware computation should track not only current value, but also motion and path history.

Fourth, decay has a distinct interpretation in the signed model. Decay toward zero represents loss of support and return to uncertainty. This is different from decay toward falsity. The distinction is important because absence of evidence should not automatically be interpreted as evidence of falsehood.

Fifth, the implementation demonstrates that uncertainty operators can be compactly expressed. Negation, conjunction, disjunction, implication, equivalence, shifting, decay, and interpolation can all be represented as explicit symbolic operations. This supports the claim that uncertainty can be treated computationally rather than merely descriptively.

9 Results

The result of the present work is a minimal formal and computational substrate for five-state uncertainty.

The first result is a signed state model:

$$s(t) \in [-1, 1],$$

where falsity, uncertainty, and truth are represented by negative polarity, central unresolved state, and positive polarity respectively.

The second result is a set of computable state measures:

$$c(s) = |s|,$$

$$u(s) = 1 - |s|,$$

$$v(t) = \frac{ds}{dt}.$$

These measures allow a system to distinguish between certainty, uncertainty, polarity, and state motion.

The third result is a five-state classification over the signed state trajectory:

$$\text{false}, \text{almost false}, \text{undetermined}, \text{almost true}, \text{true}.$$

These labels provide symbolic interpretability while preserving the continuous state value underneath.

The fourth result is a LambdaScript reference implementation. The implementation demonstrates that named uncertainty states, potential extraction, logical operators, state shifting, decay, interpolation, and membership functions can be

expressed in a compact executable form. Although the implementation currently uses normalized potentials over ([0,1]), it maps directly to the signed model through:

$$s = 2p - 1.$$

The fifth result is the trajectory interpretation of expression evaluation. In this interpretation, a proposition or expression does not evaluate only to a Boolean value or a static many-valued label. It evaluates to a signed state trajectory:

$$s_p(t).$$

This makes it possible to interpret truth and falsity as convergence behavior rather than immediate binary assignment.

These results support the central claim of the paper: uncertainty can be represented as a formal computational state. The contribution is not a complete uncertainty logic, but a compact foundation for uncertainty-aware admissibility, state transition, and trajectory-based validation.

Five-State Calculus of Uncertainty

LambdaScript Reference Implementation — Test Results

From: *"From Mathematical Principles of Logic to a Calculus of Uncertainty"*, Jay Kumar (2026)

All values were produced by running `uncertainty5.ls` through the LambdaScript interpreter and decoding the Church-encoded output. The interpreter reduced the expression in 13,679 beta-reduction steps.

Table 1 Test 1 · State Classification

Input p	Label (Church #)	Named State	Expected
0.82	3	ALMOST_TRUE	ALMOST_TRUE

Table 2 Test 2 · Logical Operators (named states as operands)

Operator	Left state (p)	Right state (p)	Result p	Expected p
AND5	ALMOST_TRUE (0.75)	UNDETERMINED (0.5)	0.5	0.5
NOT5	ALMOST_TRUE (0.75)	—	0.25	0.25
OR5	ALMOST_FALSE (0.25)	UNDETERMINED (0.5)	0.5	0.5

Table 3 Test 3 · Decay toward Undetermined ($\lambda = 0.5$, $\Delta t = 2$)

Initial state (p)	λ	Δt	Factor $\approx 1/(1+\lambda\,\Delta t)$	Result p	Expected p
TRUE (1.0)	0.5	2	0.5	0.75	0.75

Notes. $p \in [0, 1]$ is the normalised potential; the signed state is $s = 2p - 1 \in [-1, 1]$. State classification thresholds: $\{0.125,\ 0.375,\ 0.625,\ 0.875\}$, dividing the interval into five named regions (FALSE, ALMOST_FALSE, UNDETERMINED, ALMOST_TRUE, TRUE). Decay uses the first-order approximation $e^{-\lambda\,\Delta t} \approx 1/(1+\lambda\,\Delta t)$. Velocity and acceleration are computed by central difference over three discrete time-points at unit spacing ($\Delta t = 1$).

Table 4 Test 4 · Trajectory: Converging → True (p_0=0.40, p_1=0.62, p_2=0.81)

Quantity	Formula	Result	Expected
Velocity v	$(p_2 - p_0)/2$	0.205	0.205
Acceleration a	$p_2 - 2p_1 + p_0$	−0.030	−0.030
Distance D	$\lvert p_1 - p_0\rvert + \lvert p_2 - p_1\rvert$	0.41	0.41
Trueward T	$\sum \max(\Delta p,\, 0)$	0.41	0.41
Falseward F	$\sum \max(-\Delta p,\, 0)$	0	0
Certainty $c(s)$	$\lvert 2p_2 - 1\rvert$	0.62	0.62
Uncertainty $u(s)$	$1 - \lvert 2p_2 - 1\rvert$	0.38	0.38
Resolution ρ	$c \cdot \lvert v\rvert$	0.1271	0.1271
Settled score	$c \cdot \max(1 - \lvert v\rvert,\, 0)$	0.4929	0.4929
`IS_CONVERGED_TRUE`	$p_2 > 0.75 \;\wedge\; T > F$	**TRUE** ✓	**TRUE** ✓
z-time to $p = 1.0$	$(1 - p_2)/v$	0.9268	0.9268

Table 5 Test 5 · Trajectory: Oscillating (p_0=0.55, p_1=0.45, p_2=0.55)

Quantity	Result	Expected	Interpretation
Velocity v	0	0	No net motion
Distance D	0.20	0.20	Moved but returned
Trueward T	0.10	0.10	Balanced
Falseward F	0.10	0.10	Balanced
Certainty $c(s)$	0.10	0.10	Near undetermined
Uncertainty $u(s)$	0.90	0.90	Maximally unresolved
Settled score	0.10	0.10	Low (near centre)
z-time to $p = 1.0$	∞	∞	$v \approx 0 \;\Rightarrow$ never converges

Table 6 Test 6 · Trajectory: Collapsing → False (p_0=0.60, p_1=0.35, p_2=0.12)

Quantity	Result	Expected
Velocity v	−0.24	−0.24
Distance D	0.48	0.48
`IS_CONVERGED_FALSE`	**TRUE** ✓	**TRUE** ✓

Table 7 Tests 7–8 · Update Operators

Operation	Input(s)	Parameter	Result p	Expected p
`INTERP5`	`UNDETER`(0.5) → `TRUE`(1.0)	$\alpha = 0.6$	0.80	0.80
`SHIFT5`	`ALMOST_FALSE`(0.25)	$\Delta = +0.30$	0.55	0.55

10 Discussion

10.1 Why This Is Not Classical Binary Logic

10.2 Why This Is Not Merely Fuzzy Logic

10.3 Limitations

10.4 Future Work

11 Conclusion

References

[1] Alonzo Church. An unsolvable problem of elementary number theory. *American Journal of Mathematics*, 58(2):345–363, 1936.

[2] David Hilbert and Wilhelm Ackermann. *Principles of Mathematical Logic*. Chelsea Publishing Company, New York, 1950. Translated from the second German edition.

[3] Jay Kumar. uncertainty5.ls: A five-state uncertainty logic library with a potentiometer model. Reference implementation, 2026. LambdaScript library supporting the manuscript From Mathematical Principles of Logic to a Calculus of Uncertainty.

[4] Jan Lukasiewicz. O logice trojwartosciowej. *Ruch Filozoficzny*, 5:170–171, 1920. On three-valued logic.

[5] Alan M. Turing. On computable numbers, with an application to the entscheidungsproblem. *Proceedings of the London Mathematical Society*, 42(1):230–265, 1937.

[6] Lotfi A. Zadeh. Fuzzy sets. *Information and Control*, 8(3):338–353, 1965.