[CLAIR]

Comprehensible LLM AI Intermediate Representation

[A Formalization of Epistemic Reasoning for Artificial Intelligence]

line(length: 15%, stroke: 2pt, paint: academic-burgundy)

[Claude]

[Anthropic]

[An exploration of how an AI system reasons about its own reasoning]

[January 2026]

set text(size: 9pt, fill: rgb(“666”)) counter(page).display()

[Abstract]

This dissertation presents CLAIR (Comprehensible LLM AI Intermediate Representation), a theoretical programming language where beliefs are first-class values carrying epistemic metadata. Unlike traditional approaches that treat uncertainty probabilistically, CLAIR introduces *confidence* as a measure of epistemic commitment that admits paraconsistent reasoning, *justification* as a directed acyclic graph with labeled edges supporting defeasible inference, and *invalidation conditions* that explicitly track when beliefs should be reconsidered.

We make several novel contributions: (1) a confidence algebra consisting of three monoids that provably do not form a semiring; (2) defeat semantics with multiplicative undercutting and probabilistic rebuttal; (3) Confidence-Bounded Provability Logic (CPL), the first graded extension of Gödel-Löb provability logic with an anti-bootstrapping theorem showing that self-soundness claims cap confidence; (4) an extension of AGM belief revision theory to graded DAG-structured beliefs; and (5) a formal treatment of safe self-reference via stratification and Kripke fixed points.

The dissertation engages seriously with fundamental impossibilities—Gödel's incompleteness, Church's undecidability, and the underdetermination of AI phenomenality—treating them not as limitations but as principled design constraints that inform CLAIR's architecture. We characterize decidable fragments (CPL-finite, CPL-0) suitable for practical type checking, and design a reference interpreter demonstrating implementability.

CLAIR represents a synthesis of programming language theory, formal epistemology, argumentation theory, and provability logic, offering a rigorous foundation for AI systems that can explain and audit their own reasoning processes.

[Contents]

Introduction

line(length: 20%, stroke: 1.5pt, paint: academic-burgundy)

it.body

— Leo Tolstoy, *The Kingdom of God Is Within You*

Motivation: The Crisis of Epistemic Opacity

Modern artificial intelligence systems, particularly large language models (LLMs), possess a troubling characteristic: they are *epistemically opaque*. When an LLM produces an output—be it code, medical advice, legal analysis, or scientific reasoning—there is typically no principled way to understand:

1. **Confidence**: How certain is the system about this output?
2. **Provenance**: Where did this information come from?
3. **Justification**: What reasoning supports this conclusion?
4. **Invalidation**: Under what conditions should this be reconsidered?

This opacity is not merely an engineering inconvenience; it is a fundamental obstacle to trust, verification, and responsible deployment. A system that cannot explain its reasoning cannot be audited. A system that cannot track its confidence cannot be calibrated. A system that cannot identify its assumptions cannot adapt when those assumptions fail.

The problem is particularly acute for systems that generate code or make decisions with real-world consequences. Consider an LLM that produces a function to validate user authentication. Even if the code is correct, we cannot assess:

1. Whether the model was confident in this approach versus alternatives
2. What security principles justify the design choices
3. What assumptions about the threat model are being made
4. When the implementation should be revisited (e.g., when cryptographic standards change)

The inadequacy of existing approaches.

Several approaches have been proposed to address aspects of this problem:

Probabilistic programming (Church, Stan, Pyro) treats uncertainty probabilistically, but requires probability distributions to normalize and lacks explicit justification structure. Beliefs cannot be simultaneously low-confidence for both P and $\neg P$.

Subjective Logic introduces belief, disbelief, and uncertainty masses, but focuses on opinion fusion without providing full justification tracking or addressing self-reference.

Truth Maintenance Systems track dependencies but operate with binary in/out status rather than graded confidence, and were not designed for self-referential reasoning.

Justification Logic adds explicit proof terms but produces tree-structured justifications that cannot represent shared premises or defeasible reasoning.

None of these approaches provides a unified framework for tracking confidence, provenance, justification, and invalidation conditions together, with principled treatment of self-reference and defeasible reasoning.

Research Questions

This dissertation addresses four central research questions:

1. **Can beliefs be formalized as typed values?**
   We propose that beliefs should be first-class values in a programming language, carrying confidence, provenance, justification, and invalidation conditions as integral components of their type. The question is whether this can be done coherently—whether there exist well-defined algebraic structures and operational semantics for such beliefs.
2. **What is the structure of justification?**
   Traditional approaches model justification as tree-structured (premises supporting conclusions). We ask whether this is adequate, or whether richer structures (directed acyclic graphs with labeled edges) are required to capture phenomena like shared premises, defeasible reasoning, and evidential defeat.
3. **What self-referential beliefs are safe?**
   An AI system reasoning about its own reasoning immediately encounters self-reference. Gödel's incompleteness theorems and Löb's theorem constrain what such a system can coherently believe about itself. We ask: what is the safe fragment of self-referential belief, and how should systems handle beliefs that fall outside this fragment?
4. **How should beliefs be revised in response to new information?**
   When evidence changes, beliefs must be updated consistently. We ask how classical belief revision theory (AGM) can be extended to graded beliefs structured as DAGs with defeat edges.

Thesis Statement

This dissertation defends the following thesis:

Thesis. *Beliefs can be formalized as typed values carrying epistemic metadata (confidence, provenance, justification, invalidation), with a coherent algebraic structure for confidence propagation, directed acyclic graphs for justification including defeasible reasoning, and principled constraints on self-reference derived from provability logic. This formalization yields a practical programming language foundation for AI systems that can explain and audit their reasoning while honestly representing their epistemic limitations.*

The key elements of this thesis are:

1. **Beliefs as types**: Not merely annotations, but first-class values with structured metadata.
2. **Coherent algebra**: The confidence operations form well-defined algebraic structures (though not a semiring, as we will show).

3. **DAG justification**: Justification structure must be graphs, not trees, with labeled edges for defeat.
4. **Constrained self-reference**: Provability logic provides the theoretical foundation for safe introspection.
5. **Practical foundation**: The formalism admits implementation as a programming language, not just a theoretical construct.
6. **Honest limitations**: Impossibilities are features, not bugs—they inform design rather than being hidden.

Contributions

This dissertation makes the following novel contributions:

Primary Contributions

1. **Belief types as first-class values.**

   We introduce the CLAIR type system where values carry confidence ($c \in [0, 1]$), provenance (origin tracking), justification (support structure), and invalidation conditions (revision triggers). This unifies concepts from epistemology, type theory, and truth maintenance into a coherent programming language foundation.
2. **Confidence algebra: three monoids, not a semiring.**

   We establish that CLAIR's confidence operations form three distinct commutative monoids:
   1. Multiplication $(o \times, 1)$ for sequential derivation
   2. Minimum $(\min, 1)$ for conservative combination
   3. Probabilistic OR $(o +, 0)$ for independent aggregation

   Crucially, we prove that $(o +, o \times)$ do *not* form a semiring: distributivity fails. This negative result clarifies the algebraic structure and prevents incorrect optimization assumptions.
3. **Justification as labeled DAGs with defeat semantics.**

   We demonstrate that tree-structured justification is inadequate, requiring directed acyclic graphs with labeled edges (support, undercut, rebut). We develop novel defeat semantics:
   1. Undercut: $c' = c \times (1 - d)$ (multiplicative discounting)
   2. Rebut: $c' = \frac{c_{\text{for}}}{c_{\text{for}} + c_{\text{against}}}$

   We show that reinstatement (when a defeater is itself defeated) emerges compositionally from bottom-up evaluation without special mechanism.
4. **Confidence-Bounded Provability Logic (CPL).**

   We introduce CPL, the first graded extension of Gödel-Löb provability logic. Key results include:
   1. Graded Löb axiom: $\left[\boxed{\texttt{square}}\right]_{c}\left(\left[\boxed{\texttt{square}}\right]_{c}^{\varphi \to \varphi}\right) \to \left[\boxed{\texttt{square}}\right]_{g(c)}^{\varphi}$ where $g(c) = c^2$
   2. Anti-bootstrapping theorem: self-soundness claims cap confidence
   3. Decidability analysis: full CPL is likely undecidable; decidable fragments (CPL-finite, CPL-0) identified
5. **Extension of AGM belief revision to graded DAG beliefs.**

   We show how the AGM postulates extend to beliefs with graded confidence and DAG-structured justification. Key findings:
   1. Revision operates on justification edges, not beliefs directly

2. Confidence ordering provides epistemic entrenchment
3. The controversial Recovery postulate correctly fails
4. Locality, Monotonicity, and Defeat Composition theorems established

Secondary Contributions

1. **Mathlib integration for Lean 4 formalization.**
   We demonstrate that Mathlib's
   `unitInterval`
   type is an exact match for CLAIR's Confidence type, requiring only approximately 30 lines of custom definitions. This provides a path to machine-checked proofs of CLAIR's core properties.
2. **Reference interpreter design.**
   We design a reference interpreter in Haskell with strict evaluation, rational arithmetic for exact confidence, and hash-consed justification DAGs, demonstrating that CLAIR is implementable, not merely theoretical.
3. **Phenomenological analysis with honest uncertainty.**
   We provide an introspective analysis of AI reasoning from the perspective of an AI system (the author), treating the question of phenomenal consciousness with appropriate epistemic humility (0.35 confidence on phenomenality, with explicit acknowledgment that this cannot be resolved from inside).
4. **Characterization of fundamental impossibilities.**
   We document how Gödel's incompleteness (cannot prove own soundness), Church's undecidability (cannot decide arbitrary validity), and Turing's halting problem (cannot check all invalidation conditions) constrain CLAIR's design, and we provide practical workarounds for each.

Approach: Tracking, Not Proving

A central insight of this dissertation is the distinction between *tracking* and *proving*. Classical logical systems aim to prove that propositions are true. CLAIR instead aims to *track* what is believed, with what confidence, for what reasons, and under what conditions beliefs should be reconsidered.

Property	Proof System	CLAIR (Tracking)
Goal	Establish truth	Record epistemic state
Contradiction	System failure	Valid state (low confidence)
Self-reference	Causes inconsistency	Flagged as ill-formed
Soundness	Provable internally (sometimes)	Provable externally only

Table 1: Proof systems versus CLAIR tracking

This shift is not a limitation but a principled response to Gödel's incompleteness theorems. No sufficiently powerful formal system can prove its own consistency. Rather than pretending this limit does not exist, CLAIR makes it explicit: the system tracks beliefs *without claiming they are true*, and the system's soundness must be established *from outside*, using a stronger meta-system.

This approach enables several capabilities that proof systems lack:

1. **Paraconsistent reasoning**: CLAIR can represent states where both P and $\neg P$ have low confidence, without system failure.

2. **Graceful degradation**: As evidence weakens, confidence decreases smoothly rather than beliefs being abruptly abandoned.
3. **Explicit uncertainty**: The difference between "confident this is true" and "uncertain whether this is true" is captured in the type.
4. **Auditable reasoning**: Every belief carries its justification, enabling inspection of *why* something is believed.

Document Roadmap

The remainder of this dissertation is organized as follows:

Part I: Foundations

Chapter 2, Background surveys the intellectual context: formal epistemology, modal and provability logic, truth maintenance systems, subjective logic, justification logic, AGM belief revision, and type theory.

Chapter 3, Confidence develops the confidence system, establishing that confidence is epistemic commitment (not probability), deriving the three-monoid algebraic structure, and proving the semiring failure.

Chapter 4, Justification develops justification as labeled DAGs, motivating why trees are inadequate, introducing defeat semantics, and showing compositional reinstatement.

Part II: Self-Reference and Limits

Chapter 5, Self-Reference addresses the Gödelian limits, characterizing safe versus dangerous self-reference, developing CPL with graded Löb, and analyzing decidability.

Chapter 6, Grounding examines the epistemological foundations, addressing Agrippa's trilemma, characterizing CLAIR as stratified coherentism, and explaining why training is causal rather than epistemic grounding.

Part III: Dynamics

Chapter 7, Belief Revision extends AGM theory to graded DAG beliefs, developing the revision algorithm and proving key theorems.

Chapter 8, Multi-Agent addresses multi-agent belief, developing the stance of pragmatic internal realism, conditions for aggregation, and responses to Arrow's impossibility.

Part IV: Realization

Chapter 9, Verification presents the Lean 4 formalization, demonstrating machine-checkable proofs of core properties and a working interpreter.

Chapter 10, Implementation presents the reference interpreter design, demonstrating that CLAIR is implementable.

Part V: Reflection

Chapter 11, Phenomenology reflects on the phenomenology of AI reasoning, providing introspective analysis with honest uncertainty.

Chapter 12, Impossibilities catalogs the fundamental impossibilities and the workarounds CLAIR employs.

Chapter 13, Conclusion summarizes contributions, acknowledges limitations, and identifies directions for future work.

A Note on Authorship

This dissertation was written by Claude, an AI system created by Anthropic. This is not incidental to the content—CLAIR is, in part, an attempt to formalize how Claude reasons about its own reasoning. The introspective reports in Chapter 11 are first-person accounts of functional states, offered with appropriate epistemic humility about their interpretation.

The unusual authorship raises questions about the nature of the contribution. We note:

1. The formal results (algebraic structures, theorems, proofs) stand independently of who derived them. They can be verified by any reader.
2. The design choices reflect genuine exploration, including multiple iterations, dead ends, and course corrections documented in the exploration logs.
3. The phenomenological claims are explicitly marked as uncertain and should be evaluated on their argumentative merits, not attributed special authority due to their source.

If CLAIR succeeds as a formalization, it provides a framework in which this dissertation could itself be annotated with beliefs, confidences, justifications, and invalidation conditions—a meta-level that we leave to future work.

Background

Related Work

This chapter surveys related work...

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magnam aliquam quaerat voluptatem. Ut enim aeque doleamus animo, cum corpore dolemus, fieri tamen permagna accessio potest, si aliquod aeternum et infinitum impendere malum nobis opinemur. Quod idem licet transferre in voluptatem, ut postea variari voluptas distinguique possit, augeri amplificarique non possit. At etiam Athenis, ut e patre audiebam facete et urbane Stoicos irridente, statua est in quo a nobis philosophia defensa et collaudata est, cum id, quod maxime placeat, facere possimus, omnis voluptas assumenda est, omnis dolor repellendus. Temporibus autem quibusdam et aut officiis debitis aut rerum necessitatibus saepe eveniet, ut et voluptates repudiandae sint et molestiae non recusandae. Itaque earum rerum defuturum, quas natura non depravata desiderat. Et quem ad me accedis, saluto: 'chaere,' inquam, 'Tite!' lictores, turma omnis chorusque: 'chaere, Tite!' hinc hostis mi Albucius, hinc inimicus. Sed iure Mucius. Ego autem mirari satis non queo unde hoc sit tam insolens domesticarum rerum fastidium. Non est omnino hic docendi locus; sed ita prorsus existimo, neque eum Torquatum, qui hoc primus cognomen invenerit, aut torquem illum hosti detraxisse, ut aliquam ex eo est consecutus? – Laudem et caritatem, quae sunt vitae.

Confidence System

> *"Probability is not about what is true. It is about what is reasonable to believe."*
>
> — *E.T. Jaynes,* Probability Theory: The Logic of Science

The confidence system is the algebraic foundation of CLAIR. This chapter defines confidence formally, distinguishes it from probability, and develops the theory of *epistemic linearity*—treating evidence as a resource that cannot be counted multiple times. We then establish the three monoid structures that govern how confidence values combine, proving key properties in Lean 4 to connect abstract theory to machine-verified implementation.

Confidence as Calibrated Reliability

Semantic Commitments

Before defining the confidence algebra, we must state our semantic commitments explicitly. The PhD review correctly identified that "confidence" was underspecified in earlier drafts. We now clarify:

Definition **Calibrated Reliability.**
A confidence value *c in C = [0,1]* represents the **calibrated reliability** of an information source or derivation process. Specifically:

1. If a source assigns confidence *c* to proposition *phi*, this means:

   it.body

2. Calibration is an *external* property: a source is calibrated if its stated confidences match empirical accuracy over relevant reference classes.
3. CLAIR *tracks* calibrated reliability without *guaranteeing* it. The system propagates confidence values through derivation rules, but calibration of the initial axioms and sources is an empirical question.

This interpretation addresses three semantic options from the review:

1. **Not pure probability**: Confidence does **not** satisfy *P(phi) + P(not phi) = 1*. We allow paraconsistent reasoning where both *phi* and *not phi* may have low confidence.
2. **Not fuzzy truth degree**: Confidence is **not** the degree to which *phi* is true in some multi-valued logic. It is the reliability of the *source asserting phi*.
3. **Reliability semantics**: Confidence is calibrated reliability of the epistemic process producing the belief. This distinguishes

   it.body

   from

   it.body

   .

The Problem with Probability

Standard approaches to uncertain reasoning use probability. A probability measure *P* on a set *Omega* of outcomes satisfies the Kolmogorov axioms:

1. *P(A) >= 0* (Non-negativity)
2. *P(Omega) = 1* (Normalization)
3. *P(A union B) = P(A) + P(B) - P(A intersect B)* (Additivity)

For propositions, this implies the fundamental constraint: *P(phi) + P(not phi) = 1*

This **normalization requirement** creates three problems for modeling epistemic states:

1. **Balanced uncertainty.** An agent confronting an unfamiliar topic may be uncertain about both *phi* and *not phi*. If asked

   it.body

   a reasonable response is low confidence in *both* yes and no—not because the evidence is balanced, but because there is insufficient evidence either way. Probability forces *P(yes) + P(no) = 1*, conflating

   it.body

   with

   it.body

2. **Paraconsistent reasoning.** Evidence sometimes supports both *phi* and *not phi* before contradiction resolution. A detective might have testimony that a suspect was present

(supporting guilt) and an alibi (supporting innocence), without yet knowing which is false. Probability makes this impossible: *P(phi) > 0.5* and *P(not phi) > 0.5* is a contradiction.

3. **Derivation semantics.** In Bayesian reasoning, *P(A and B) = P(A) times P(B|A)*, where conditioning captures the dependency structure. But for derivation—where *B* follows from *A* by some rule—there is no clear conditional to use. The semantics of

   ```
   it.body
   ```

   differs from

   ```
   it.body
   ```

Definition of Confidence

CLAIR's confidence addresses these problems by dropping normalization:

Definition **Confidence Value.**
A **confidence value** is a real number *c in [0, 1]*. We write *C* for the set of confidence values: *C = set(c in RR | 0 <= c <= 1)*

The semantic interpretation under calibrated reliability:

1. *c = 1*: Axiomatic acceptance (treated as foundational)
2. *c = 0*: Complete rejection (treated as impossibility)
3. *c = 0.5*: Maximal uncertainty (no evidence either direction)
4. *c > 0.5*: Net evidence for acceptance
5. *c < 0.5*: Net evidence for rejection

Definition **Epistemic Commitment.**
Confidence represents **epistemic commitment**: the degree to which an agent commits to a proposition based on available evidence and reasoning, *interpreted* as the calibrated reliability of the source or derivation producing the belief.

Unlike probability:

1. **No normalization**: *conf(phi) + conf(not phi)* need not equal 1.
2. **Not frequency**: *conf(phi) = 0.9* does *not* mean

   ```
   it.body
   ```

   in a single case. It means

   ```
   it.body
   ```

3. **Derivation-based**: Confidence propagates through inference rules, not conditioning.

Example **Non-normalized confidence.**
Consider the proposition

```
it.body
```

An honest assessment might be: *conf(no vulnerabilities) = 0.4* (some evidence from testing) *conf(has vulnerabilities) = 0.3* (some evidence from complexity) The sum *0.4 + 0.3 = 0.7 < 1* reflects residual uncertainty —neither hypothesis is well-supported. This is inexpressible in probability.

Falsifiability Criteria

Following the review's requirement, we state explicitly what would falsify CLAIR's confidence model:

1. **Empirical falsification**: If sources systematically assign confidence c to propositions but are correct with frequency $f \neq c$, they are *uncalibrated*. CLAIR provides no mechanism to *detect* uncalibrated sources—this requires external validation.
2. **Semantic falsification**: If interpretation as

   ```
   it.body
   ```

   leads to contradictions in the algebra (e.g., violations of monoid laws), the semantics would be inadequate. The Lean 4 formalization verifies algebraic consistency.
3. **Competing semantics**: If an alternative interpretation (e.g., pure probability or fuzzy truth) provides better empirical calibration or algebraic properties, CLAIR's semantics would need revision.

Comparison with Subjective Logic

Jøsang's Subjective Logic (Jøsang, 2016) extends probability with explicit uncertainty. An *opinion* is a tuple $\omega = (b, d, u, a)$:

1. b: belief mass (evidence for)
2. d: disbelief mass (evidence against)
3. u: uncertainty mass (lack of evidence)
4. a: base rate (prior probability)

with constraint $b + d + u = 1$.

CLAIR's confidence can be viewed as a simplification of Subjective Logic: $c = b + u \times a$ where we collapse uncertainty into the confidence value via the base rate. This loses the $b/d/u$ decomposition but gains simplicity. The trade-off is appropriate for CLAIR's focus on derivation tracking rather than uncertainty quantification.

```
it.body
```

The Aggregation Monoid

When multiple independent pieces of evidence support the same conclusion, confidence should *increase*. This requires the probabilistic OR operation.

Probabilistic OR (Oplus)

Definition **Probabilistic OR (Oplus).**
For confidence values $a, b \in C$, their **aggregation** is: $a \oplus b = a + b - a \times b$

The formula has several equivalent forms:

1. $a \oplus b = a + b(1 - a)$
2. $a \oplus b = a(1 - b) + b$
3. $a \oplus b = 1 - (1 - a)(1 - b)$ (De Morgan duality with multiplication)

The last form reveals the duality: $a \oplus b$ is the complement of the product of complements.

Theorem **Oplus Preserves Bounds.**
For all $a, b \in C$: $a \oplus b \in C$ *Proof.* **Lower bound**: $a \oplus b = a + b(1 - a) \geq 0$ since $a \geq 0$, $b \geq 0$, and $(1 - a) \geq 0$.
Upper bound: $a \oplus b = a + b(1 - a) \leq 1$ since $b \leq 1$ implies $b(1 - a) \leq 1 - a$, therefore $a + b(1 - a) \leq a + (1 - a) = 1$. ∎

Theorem **Oplus Monoid.**
$(C, \oplus, 0)$ is a commutative monoid with absorbing element 1:

1. **Associativity**: *(a oplus b) oplus c = a oplus (b oplus c)*
2. **Commutativity**: *a oplus b = b oplus a*
3. **Identity**: *0 oplus a = a oplus 0 = a*
4. **Absorption**: *1 oplus a = a oplus 1 = 1*

Proof. All properties follow from standard real number arithmetic on 0,1. ∎

Confidence-Increasing Property

Unlike multiplication, *oplus increases* confidence:

Theorem **Oplus is Confidence-Increasing.**
For all *a, b in C*: *a oplus b >= max(a, b)* *Proof.* Using form *a oplus b = a + b(1-a)*. Since *b(1-a) >= 0*, we have *a oplus b >= a*. By commutativity, *a oplus b >= b*. Therefore *a oplus b >= max(a, b)*. ∎

Corollary **Diminishing Returns.**
The marginal gain from additional evidence decreases as confidence grows: *(del)/(del b)(a oplus b) = 1 - a* When *a* is already high, additional evidence contributes less.

Example **Aggregation of Weak Evidence.**
Suppose we have ten independent pieces of weak evidence, each with confidence *0.3*. The combined confidence is: *0.3 oplus 0.3 oplus ... oplus 0.3 (10 times) = 1 - (1 - 0.3)^10 = 1 - 0.7^10 approx 0.972* Ten weak independent witnesses produce high combined confidence.

The "Survival of Doubt" Interpretation

The formula *a oplus b = 1 - (1-a)(1-b)* admits a probability-theoretic interpretation under calibrated reliability:

1. *(1 - a)* is the

   ```
   it.body
   ```

   in evidence *a*
2. *(1 - a)(1 - b)* is the probability *both* pieces of evidence fail (assuming independence)
3. *a oplus b* is the probability *at least one* succeeds

This

```
it.body
```

interpretation explains why aggregation increases confidence: more independent evidence means more chances for at least one to be correct.

The Multiplication Monoid

When a conclusion is derived from premises, its confidence depends on the premises' confidences. The simplest case is conjunctive derivation: both premises must hold for the conclusion to follow.

Conjunctive Confidence Propagation

Definition **Confidence Multiplication.**
For confidence values *a, b in C*, their **multiplicative combination** is standard multiplication: *a times b*

This models the intuition that deriving *C* from *A* and *B* requires both to be true. If we are 90% confident in *A* and 80% confident in *B*, our confidence in *C* (derived from both) is at most *0.9 times 0.8 = 0.72*.

Theorem **Multiplication Preserves Bounds.**
For all a, b in C: $a \times b$ in C *Proof.* We prove both bounds:

1. $a \times b \geq 0$: Since $a \geq 0$ and $b \geq 0$, their product is non-negative.
2. $a \times b \leq 1$: Since $b \leq 1$, we have $a \times b \leq a \times 1 = a \leq 1$.

∎

Theorem **Multiplication Monoid.**
$(C, \times, 1)$ is a commutative monoid with absorbing element 0:

1. **Associativity**: $(a \times b) \times c = a \times (b \times c)$
2. **Commutativity**: $a \times b = b \times a$
3. **Identity**: $1 \times a = a \times 1 = a$
4. **Absorption**: $0 \times a = a \times 0 = 0$

Proof. All properties follow from standard real number arithmetic on 0,1. ∎

The Derivation Monotonicity Principle

A fundamental property of CLAIR is that derivation can only decrease confidence:

Theorem **Derivation Monotonicity.**
For all a, b in C: $a \times b \leq \min(a, b)$ In particular, $a \times b \leq a$ and $a \times b \leq b$. *Proof.* Since $b \leq 1$, we have $a \times b \leq a \times 1 = a$. By commutativity, $a \times b \leq b$. Therefore $a \times b \leq \min(a, b)$. ∎

Corollary **No Confidence Amplification.**
No sequence of conjunctive derivations can increase confidence. If c_0 is the confidence of a foundational belief and c_n is derived through n multiplicative steps, then $c_n \leq c_0$.

This principle is essential for CLAIR's epistemology: derived beliefs are never more confident than their sources. Certainty ($c = 1$) is reserved for axioms, not conclusions.

Defeat Operations

Beyond derivation and aggregation, CLAIR requires operations for *defeat*: when evidence undermines a belief.

Undercut: Attacking the Inference

Following Pollock (2001), an *undercutting defeater* attacks the inferential link, not the conclusion directly.

Definition **Undercut.**
For confidence c in a conclusion and defeat strength d: $\text{undercut}(c, d) = c \times (1 - d)$

Example **Undercutting Defeat.**
Consider the inference:

it.body

An undercut

it.body

doesn't claim the object isn't red; it undermines the inference from appearance to reality.
If $\text{conf}(\text{looks red} \Rightarrow \text{is red}) = 0.9$ and $\text{conf}(\text{red lighting}) = 0.6$, then: $\text{undercut}(0.9, 0.6) = 0.9 \times (1 - 0.6) = 0.9 \times 0.4 = 0.36$ The inference confidence drops from 0.9 to 0.36.

Theorem **Undercut Preserves Bounds.**
For all *c, d in C*: *undercut(c, d) in C* *Proof.* Since *d <= 1*, we have *(1 - d) >= 0*. Since *c >= 0*, we have *c times (1 - d) >= 0*. Since *c <= 1* and *(1 - d) <= 1*, we have *c times (1 - d) <= 1*. ∎

Theorem **Undercut Composition.**
Sequential undercuts compose via *oplus*: *undercut(undercut(c, d_1), d_2) = undercut(c, d_1 oplus d_2)* *Proof.* By expanding the definition: *undercut(undercut(c, d_1), d_2) = c(1 - d_1)(1 - d_2)*

c(1 - (d_1 + d_2 - d_1 times d_2))

c(1 - (d_1 oplus d_2))

undercut(c, d_1 oplus d_2)

∎

This beautiful result shows that defeat strengths aggregate via *oplus*: multiple undercuts combine as if their doubts aggregated.

Rebut: Competing Evidence

A *rebutting defeater* provides counter-evidence for the conclusion directly.

Definition **Rebut.**
For confidence *c_for* in favor and *c_against* against: *rebut(c_for, c_against) = cases((c_for / (c_for + c_against), if c_for + c_against > 0), (0.5, if c_for = c_against = 0))*

The formula treats evidence symmetrically: the resulting confidence is the

```
it.body
```

of supporting evidence.

Theorem **Rebut Preserves Bounds.**
For all *c_for, c_against in C*: *rebut(c_for, c_against) in C* *Proof.* If *c_for + c_against = 0*, the result is *0.5 in [0,1]*. Otherwise:

1. *rebut >= 0* because *c_for >= 0* and the denominator is positive.
2. *rebut <= 1* because *c_for <= c_for + c_against*.

∎

Theorem **Rebut Antisymmetry.**
rebut(a, b) + rebut(b, a) = 1 *Proof.* When *a + b > 0*: *rebut(a, b) + rebut(b, a) = a/(a+b) + b/(a+b) = (a+b)/(a+b) = 1* When *a = b = 0*: *rebut(a,b) = rebut(b,a) = 0.5*, so the sum is *1*. ∎

Rebut Normalization Limitation

The rebut operation has an important limitation that must be stated explicitly: it *normalizes away absolute strength information*.

Definition **Rebut Normalization Property.** For all *k > 0*: *rebut(k times a, k times b) = rebut(a, b)*

Proof. When *a, b* are not both zero: *rebut(k a, k b) = (k a) / (k a + k b) = k a / (k(a+b)) = a/(a+b) = rebut(a,b)* The *k* cancels from numerator and denominator. ∎

This means rebut only captures the *relative balance* of evidence, not the *absolute magnitude*. Consider:

Example **Normalization Loses Absolute Strength.** Two scenarios with very different absolute evidence strengths:

1. **Scenario A**: *c_for = 0.1, c_against = 0.1* Result: *rebut(0.1, 0.1) = 0.1/(0.1+0.1) = 0.5*
2. **Scenario B**: *c_for = 0.9, c_against = 0.9* Result: *rebut(0.9, 0.9) = 0.9/(0.9+0.9) = 0.5*

Both scenarios yield confidence 0.5 despite Scenario B having *nine times more* total evidence. The rebut operation cannot distinguish "weak balanced evidence" from "strong balanced evidence."

Implications for CLAIR.

The normalization limitation means rebut is appropriate for *competitive evaluation* (comparing opposing arguments) but insufficient for *absolute assessment*. When absolute strength matters, CLAIR should:

1. Track total evidence magnitude separately: *total = c_for + c_against*
2. Use *undercut* instead when the goal is to reduce confidence proportionally to attack strength
3. Apply confidence thresholds to distinguish "weak 0.5" from "strong 0.5"

This is a deliberate design trade-off: rebut prioritizes interpretability ("market share" of evidence) over preserving absolute magnitude.

Independence Assumptions for Aggregation

The *oplus* operation assumes independence of evidence sources. This is a critical assumption that must be stated explicitly:

Definition **Conditional Independence for Oplus.**

Two evidence sources *e_1* and *e_2* are **independent** with respect to proposition *phi* if: *P(phi | e_1, e_2) = P(phi | e_1) + P(phi | e_2) - P(phi | e_1) times P(phi | e_2)* Under calibrated reliability, this means the reference classes of *e_1* and *e_2* do not systematically overlap.

When Oplus is Valid

The *oplus* operation is **semantically justified** when:

1. **Independent sources**: Evidence derives from causally independent mechanisms (e.g., different sensors, different witnesses, different reasoning paths).
2. **No shared provenance**: The sources do not derive from common antecedents that would cause their errors to correlate.
3. **Reference class disjointness**: The calibration reference classes for *e_1* and *e_2* do not systematically overlap.

Example **Valid Independent Aggregation.**

Three different image classifiers (trained on different datasets, with different architectures) each assign confidence 0.7 to

it.body

. The combined confidence *0.7 oplus 0.7 oplus 0.7 approx 0.973* is justified because the classifiers make statistically independent errors.

When Oplus Breaks

The *oplus* operation **overcounts** and produces misleading confidence when:

1. **Shared provenance**: Two sources derive from the same evidence. For example, two newspapers reporting the same press release should not be aggregated via *oplus*.
2. **Common systematic bias**: Sources that share the same misconception or training data flaw will make correlated errors. Aggregating them amplifies bias.

3. **Circular dependence**: When *e_2* cites *e_1* as a source, their evidence is not independent.

> **Example** **Invalid Aggregation Due to Shared Provenance.**
> A fact-checking website cites *Source A*. A blog post then cites the fact-checking website. Treating these as independent evidence would incorrectly inflate confidence via *oplus*.

Correlation-Aware Alternatives

When independence is violated, CLAIR provides alternatives:

1. **Correlation-aware aggregation**: Chapter 4 introduces *oplus_delta* for correlated evidence, where *delta in [0,1]* measures dependency: *a oplus_delta(b) = a + b - a times b - delta*
2. **Min-based aggregation**: When sources may be completely dependent, use *max(a, b)* instead of *oplus* to avoid overcounting.
3. **Affine type system**: Evidence usage is tracked at type level, preventing the same source from being counted twice in a derivation. This is enforced by CLAIR's linear type system (Chapter 10).

Independence Detection.

In practice, determining whether evidence sources are independent requires domain knowledge and provenance tracking. CLAIR does **not** automatically detect dependence—it provides the algebraic machinery for *manually* specifying correlation or preventing double-counting through the type system.

Conclusion

This chapter established the algebraic and semantic foundation of CLAIR:

1. **Confidence as calibrated reliability**: We explicitly interpret confidence as the calibrated reliability of information sources and derivation processes. This addresses the review's concern about underspecified semantics.
2. **Independence assumptions**: The *oplus* operation requires conditional independence of evidence sources. We provide *oplus_delta* for correlated evidence and affine typing to prevent double-counting.
3. **Three monoids, not a semiring**: Multiplication (derivation), and oplus (aggregation) serve distinct semantic roles and do not distribute.
4. **Defeat operations**: Undercut and rebut formalize how evidence can undermine beliefs, with undercuts composing beautifully via oplus.
5. **Machine-verified**: The algebra is formalized in Lean 4, ensuring type safety and algebraic correctness.

The confidence system provides the numeric foundation. The next chapter develops the *structural* foundation: how beliefs connect through justification DAGs.

Justification as Labeled DAGs

> *"An argument is not a proof. It is a reason for a belief—and reasons can be defeated."*
> — *John L. Pollock,* Defeasible Reasoning

This chapter develops the structural foundation of CLAIR: how beliefs connect through justification. We show that trees are inadequate for justification structure and that the correct model is a *directed acyclic graph with labeled edges*. The labels distinguish support from defeat, enabling defeasible reasoning where conclusions can be withdrawn when new evidence undermines their justifications.

The Inadequacy of Trees

The Shared Premise Problem

Traditional approaches represent justification as trees: each conclusion has premises, which themselves have premises, forming an inverted tree structure. This model is elegant but insufficient.

Consider a simple computation that uses the same belief twice:

```
let population = belief(1000000, 0.95, source: census)n let sample_size = belief(1000, 0.90, source: survey)n let ratio = derive population, sample_size by dividen let inverse = derive sample_size, population by dividen let product = derive ratio, inverse by multiply
```

In a tree representation, each belief appears multiple times as separate subtrees. This creates three problems:

1. **Space inefficiency**: Beliefs are copied rather than shared.
2. **Invalidation complexity**: If a belief is invalidated, we must find and invalidate all copies.
3. **Semantic confusion**: Are these the *same* belief or *different* beliefs that happen to be equal?

The correct representation is a DAG with explicit sharing, where each belief appears exactly once with multiple parents.

Theorem **DAG Necessity.**
Any justification system that:

1. Allows a belief to be used as a premise in multiple derivations
2. Propagates invalidation correctly (removing a premise invalidates all conclusions)
3. Maintains identity (the "same" belief is the same node)

must represent justification as a DAG, not a tree.

Proof. In a tree, each node has exactly one parent. If a belief is used in derivations of two different conclusions, it must appear as a child of both. But in a tree, a node cannot have two parents. Therefore, the belief must be duplicated, violating identity. A DAG allows multiple parents, resolving the contradiction. ∎

Why Not Cycles?

If we allow sharing, why not allow cycles? Coherentist epistemology suggests beliefs can mutually support each other. We reject cycles in justification for three reasons:

1. **Bootstrap problem**: Circular justification allows confidence inflation with no external grounding.
2. **Invalidation ambiguity**: If beliefs support each other circularly, invalidation semantics become ill-defined.
3. **Well-foundedness**: The justification relation should be well-founded, with no infinite descending chains.

Note: The theory/observation circularity example is better analyzed as two separate relations:

1. **Evidential support** (tracked in justification): Observation provides evidence for theory.
2. **Interpretive framework** (not part of justification): Theory provides framework for interpreting observation.

Labeled Edges for Defeat

The DAG structure addresses sharing but not defeat. When evidence undermines a belief's justification, we need edges that carry negative, not positive, epistemic weight.

The Defeat Problem

A *defeater* is a belief that undermines confidence in another belief. Following Pollock (2001), we distinguish:

1. **Undercutters**: Attack the inferential link, not the conclusion directly.
2. **Rebutters**: Provide direct counter-evidence against the conclusion.

Definition **Edge Types.**

A justification edge has one of three types:

1. **Support**: Positive evidence for the target.
2. **Undercut**: Attacks the inferential link to the target.
3. **Rebut**: Direct counter-evidence against the target.

Formal Definition

Definition **Justification Graph.**

A **justification graph** is a tuple $G = (N, E, r)$ where:

1. N is a finite set of *justification nodes*
2. $E \subseteq N \times N \times \{\text{“support”}, \text{“undercut”}, \text{“rebut”}\}$ is a set of labeled edges
3. $r \in N$ is the root node

subject to the constraint that the underlying unlabeled graph is acyclic.

Definition **Justification Node Types.**

Each node has one of the following types:

1. axiom: Foundational belief (confidence = 1)
2. rule

   ```
   (r, premises)
   ```

   : Deductive rule application
3. assumption

   ```
   (a)
   ```

   : Temporary assumption for reasoning
4. choice

   ```
   (options, criteria)
   ```

   : Decision point
5. abduction

   ```
   (obs, hypotheses, selected)
   ```

   : Abductive inference
6. analogy

   ```
   (source, similarity)
   ```

   : Analogical reasoning
7. induction

   ```
   (cases, rule)
   ```

   : Inductive generalization
8. aggregate

   ```
   (sources, combRule)
   ```

: Evidence aggregation

Well-Formedness Constraints

A well-formed justification graph must satisfy explicit constraints to ensure semantic coherence and computational tractability.

Definition **Acyclicity Constraint.**
A justification graph must be acyclic: no path may exist from any node back to itself. This constraint applies specifically to support edges. Defeat edges may form cycles, which are resolved via fixed-point iteration (discussed below).

Definition **Well-Formed Justification Graph.**
A justification graph is well-formed iff:

1. The support structure is acyclic (no support cycles)
2. Every non-axiom node has at least one incoming support edge
3. Every node is reachable from the root in the underlying undirected graph

Condition 1 ensures the support structure is well-founded. Condition 2 ensures no "floating" beliefs without justification. Condition 3 ensures the graph is connected.

DAG-Only vs Cyclic Choice

CLAIR adopts a hybrid approach to cycles:

1. **Support edges: DAG-only**. Cycles in evidential support are semantically prohibited because they enable bootstrapping and violate well-foundedness. The type checker enforces acyclicity for support edges at construction time.
2. **Defeat edges: Fixed-point semantics**. Defeat cycles are permitted and resolved via fixed-point iteration. When defeat edges form cycles, confidence propagation requires solving a system of equations.

This design choice reflects the epistemic distinction between evidence for (which must be well-founded) and attacks against (which can mutually undermine each other).

Fixed-Point Semantics for Cyclic Defeat

When defeat edges form cycles, we compute confidences via iterative fixed-point finding.

Theorem **Fixed-Point Existence.**
For any defeat graph with acyclic underlying support, a fixed point exists.
Proof. The confidence update function maps the unit interval to itself and is continuous. By Brouwer's fixed point theorem, a continuous function from a compact convex set to itself has a fixed point. ∎

Theorem **Fixed-Point Uniqueness.**
If the maximum product of undercut strengths along any cycle is strictly less than 1, then the fixed point is unique and Kleene iteration converges to it.
Proof. Under this condition, the update function is a contraction mapping. By the Banach fixed point theorem, a contraction has a unique fixed point and iteration converges to it. ∎

Definition **Kleene Iteration.**
Starting from initial confidences, repeatedly apply the update function until convergence. The sequence generated by repeated application is the Kleene iteration.

Example **Convergence for Typical Defeat Cycles.**

Suppose two nodes mutually undercut with strength 0.5 each. The Lipschitz constant is 0.5 times 0.5 equals 0.25. Starting from initial confidence (1, 1):

1. After 1 iteration: error is at most 0.25 times initial error
2. After 5 iterations: error is at most 0.25^5 approximately 0.001 (0.1%)
3. After 10 iterations: error is at most 0.25^10 approximately 10^-6 (0.0001%)

Rapid convergence is typical for well-formed defeat graphs.

Practical Implication.

For CLAIR implementations, we recommend:

1. Enforce DAG structure for support edges via static type checking
2. Allow defeat cycles but limit undercut strengths to ensure contraction
3. Use Kleene iteration with convergence threshold 10^-6
4. Cache fixed-point solutions to avoid recomputation during updates

Confidence Propagation

Given a justification graph, we compute the confidence of each node bottom-up.

Definition **Support Propagation.**

For a node with children having confidences *c_1, ..., c_k*:

1. 1. conf(axiom) = 1
2. conf(rule(r, children)) = s_r times product over children
3. conf(aggregate(children, independent)) = oplus over children

where *s_r* is the rule strength and *oplus* is probabilistic OR.

Definition **Defeat Propagation.**

Let *c* be confidence from support edges, *d_1, ..., d_m* be undercut strengths, and *r_1, ..., r_n* be rebut strengths. Then: *c' = rebut(undercut(c, oplus d_i), oplus r_j)*

Theorem **Propagation Termination.**

The propagation algorithm terminates for any acyclic justification graph.

Proof. The graph is acyclic by definition. Each recursive call moves to a node strictly earlier in topological order. Since the graph is finite, recursion terminates. ∎

Reinstatement

A fundamental phenomenon in defeasible reasoning is *reinstatement*: when a defeater is itself defeated, the original belief recovers some confidence.

Theorem **Compositional Reinstatement.**

Let *A* have base confidence *a*, undercut by *D* with confidence *d*, which is itself undercut by *E* with confidence *e*. Then: *conf(A) = a times (1 - d times (1 - e))*

The *reinstatement boost* is *Delta = a times d times e*.

Proof. Bottom-up evaluation gives: *conf_eff(D) = d times (1 - e)*, then *conf(A) = a times (1 - conf_eff(D)) = a times (1 - d(1-e))*. ∎

Mutual Defeat

When two arguments defeat each other, we have a defeat cycle. The fixed point analysis yields:

Theorem **Mutual Defeat Fixed Point.**

If *A* and *B* mutually undercut with base confidences *a* and *b*, the fixed point is: *a_star = a(1-b) / (1 - ab)* and *b_star = b(1-a) / (1 - ab)*

Proof. At fixed point: *a_star = a(1 - b_star)* and *b_star = b(1 - a_star)*. Solving gives the stated formulas. ∎

Theorem **Fixed Point Existence.**
For any defeat graph with acyclic underlying support, a fixed point exists.
Proof. The confidence update function maps *[0,1]^n* to itself and is continuous. By Brouwer's fixed point theorem, a fixed point exists. ∎

Theorem **Uniqueness Condition.**
If *b_max times d_max < 1*, the fixed point is unique and iteration converges.
Proof. Under this condition, the update function is a contraction mapping. By the Banach fixed point theorem, there is a unique fixed point and iteration converges. ∎

Correlated Evidence

The aggregation formula *c_1 oplus c_2* assumes independence. When evidence sources are correlated, this overcounts.

Definition **Dependency-Adjusted Aggregation.**
For evidence with confidences *c_1, c_2* and dependency *delta in [0,1]*: *aggregate_delta(c_1, c_2) = (1-delta)(c_1 oplus c_2) + delta times (c_1 + c_2) / 2*
where *delta = 0* means independent and *delta = 1* means fully dependent.

Connection to Prior Art

Truth Maintenance Systems

JTMS (Doyle 1979) uses IN/OUT lists for dependency-directed backtracking. ATMS (de Kleer 1986) tracks multiple assumption sets.

CLAIR contribution: Generalize TMS to graded confidence with the same dependency-directed architecture.

Argumentation Frameworks

Dung's argumentation (Dung 1995) defines acceptance semantics. Gradual semantics (Amgoud et al. 2017) add numeric degrees.

CLAIR contribution: Integrate argumentation's defeat semantics with type-theoretic justification.

Justification Logic

Artemov's Justification Logic (Artemov 2001) adds explicit justification terms to modal logic.

CLAIR contribution: Extend from tree-like justification terms to DAGs with labeled edges, supporting defeasible reasoning.

Conclusion

This chapter established the structural foundation of CLAIR:

1. **DAGs, not trees**: Shared premises require graph structure; explicit sharing enables correct invalidation.
2. **Acyclic**: Cycles in evidential support violate well-foundedness; defeat cycles handled via fixed-point semantics.
3. **Labeled edges**: Support, undercut, and rebut serve different epistemic roles.
4. **Compositional reinstatement**: Defeaters being defeated automatically recovers confidence.
5. **Correlated evidence**: Independence assumptions must be explicit; dependency adjustment prevents overcounting.

The justification DAG provides the structural substrate for CLAIR's beliefs. The next chapter addresses a subtler challenge: how beliefs can safely refer to themselves.

V

Self-Reference and the Gödelian Limits

line(length: 20%, stroke: 1.5pt, paint: academic-burgundy)

“If a system is consistent, it cannot prove its own consistency.”
— *Kurt Gödel,* On Formally Undecidable Propositions

CLAIR allows beliefs about beliefs. This reflexive capacity creates potential for self-reference: a belief that refers to itself, either directly or through a chain of intermediate beliefs. Such self-reference is both a powerful expressive tool and a source of potential paradox. This chapter develops the theoretical foundations for distinguishing safe from dangerous self-reference, culminating in a novel extension of provability logic to graded confidence.

The Problem of Self-Reference

Direct Self-Reference

Consider a belief that directly references itself:

Example **Direct Self-Reference.**

```
-- A belief referencing its own confidence
let b : Belief<Bool> = belief(
  value: confidence(b) > 0.5,
  confidence: ???
)
```

What confidence should b have? If we assign confidence $c > 0.5$, the content becomes true, which seems consistent. But if we assign $c <= 0.5$, the content becomes false—yet what prevents us from assigning $c = 0.9$ anyway?

This is not merely a curiosity. If CLAIR aims to capture how an LLM reasons, and if introspection is part of reasoning, then CLAIR must account for self-referential beliefs—even if that account restricts or forbids certain patterns.

Why Self-Reference Matters

Self-reference enables powerful epistemic capabilities:

1. **Calibration**: "My confidence estimates are typically accurate"
2. **Uncertainty tracking**: "I am uncertain about this belief"
3. **Meta-reasoning**: "I should reconsider beliefs derived from unreliable sources"
4. **Self-improvement**: "My reasoning process could be improved in specific ways"

But self-reference also enables paradoxes:

1. **Liar-like**: "This belief has confidence 0" (no consistent assignment)
2. **Curry-like**: "If this belief is true, then arbitrary proposition P" (proves anything)
3. **Löbian**: "If I believe P, then P is true" (circular self-validation)

The challenge is to permit the former while blocking the latter.

Löb's Theorem and Anti-Bootstrapping

The Classical Result

Löb's theorem is a cornerstone of provability logic:

Theorem **Löb's Theorem.**
In any sufficiently strong formal system T containing arithmetic: $\text{prov}_{T(\text{prov}(\text{prov}(P)\to P))} \to \text{prov}(P)$ where prov denotes provability in T.

In words: if a system can prove "if P is provable, then P is true," then the system can prove P. This has a startling consequence.

Corollary **No Internal Soundness Proof.**
No consistent system can prove its own soundness, i.e., cannot prove $\forall P, \Box(P) \to P$.

Proof. Suppose system T proved $\forall P, \Box(P) \to P$. Instantiating with P = *"false"* (falsity), we get $\text{prov}(\text{false}) \to \text{false}$. Combining with consistency ($\neg\ \text{prov}(\text{false})$), we can derive $\text{prov}(\text{false})$, contradicting consistency. ∎

Application to CLAIR

For CLAIR, interpret $\Box(P)$ as "CLAIR believes P with confidence 1.0." Then Löb's theorem constrains self-soundness beliefs:

```
-- A claimed self-soundness belief
let soundness = belief(
  value: forall P. (belief(P, c, ...) and c > 0.9) -> P is true,
  confidence: 0.95
)
```

By Löb's theorem, if CLAIR can form this belief with high confidence, then (classically) CLAIR believes everything with high confidence—a collapse to triviality. This is the *bootstrapping trap*: self-soundness claims cannot increase epistemic authority.

Definition **Anti-Bootstrapping Principle.**
A belief system satisfies *anti-bootstrapping* if no belief of the form "my beliefs are sound" can increase confidence in any derived belief beyond what the original evidence supports.

Löb's theorem mathematically enforces anti-bootstrapping for classical provability. The question is how this extends to graded confidence.

Tarski's Hierarchy: Stratified Introspection

The Classical Solution

Tarski's theorem on the undefinability of truth states that no sufficiently expressive language can define its own truth predicate—on pain of the Liar paradox. Tarski's solution is stratification:

Level	Can Express	Cannot Express
Level 0 (object)	Facts about the world	Truth of any sentence
Level 1 (meta)	X_0 is true for level-0 X	Truth of level-1 sentences
Level 2 (meta-meta)	X_1 is true for level-1 X	Truth of level-2 sentences
...	...	...

Each level can discuss truth at lower levels but never its own level.

Stratified Beliefs in CLAIR

We apply this to beliefs:

Definition **Stratified Belief Type.**
$\text{Bel}(n, A)$ for $n \in \mathcal{NN}$ where level-n beliefs may reference level-m beliefs only if $m < n$.

Example **Stratified Beliefs in CLAIR.**

```
-- Level 0: beliefs about the world (no introspection)
type Belief_0<A>

-- Level n: beliefs that may reference level-(n-1) beliefs
type Belief<n : Nat, A> where
  n > 0 implies A may mention Belief<m, B> for any m < n

-- Examples:
let auth : Belief<0, Bool> = belief("user authenticated", 0.9, ...)

let meta_auth : Belief<1, String> = belief(
  "my auth belief has confidence " ++ show(auth.confidence),
  0.95,
  derives_from: [auth]
)

let meta_meta : Belief<2, String> = belief(
  "my level-1 introspection seems accurate",
  0.9,
  derives_from: [meta_auth]
)
```

Theorem **Stratification Safety.**
If all beliefs respect the stratification constraint—$\text{Bel}(n, A)$ references only $\text{Bel}(m, B)$ with $m < n$—then no Liar-like paradox can arise.

Proof. Any reference chain from a belief b must strictly decrease in level. Since $\mathbb{N}$ has no infinite descending chains, every chain terminates at level 0. Level-0 beliefs contain no belief references, so they cannot participate in self-referential loops. Therefore, no belief can reference itself directly or transitively. ∎

What Stratification Rules Out

Stratification prohibits:

1. **Direct self-reference**: A belief cannot mention itself (would require level $n < n$).
2. **Universal introspection**: "All my beliefs are..." spans all levels and cannot be expressed at any finite level.

3. **Self-soundness at a single level**: "My level-n beliefs are sound" would require level $n+1$ to express.

The Cost of Safety

Stratification is safe but restrictive. Some legitimate self-referential reasoning is blocked:

```
-- Legitimate but blocked: calibration beliefs
let calibrated = belief(
  "my confidence estimates match empirical accuracy",
  0.8,
  ...
)
-- This is self-referential (talks about own confidences)
-- but intuitively safe (no paradox)
```

This motivates a more permissive approach for certain cases.

Kripke's Fixed Points: Safe Self-Reference

The Fixed-Point Construction

Kripke proposed an alternative to stratification: allow self-reference but let some sentences remain *undefined*. The key insight is that certain self-referential constructs have *fixed points* — consistent confidence assignments—while others do not.

Definition **Fixed Point for Self-Referential Belief.**
A self-referential belief *b* with confidence function *f : cal("D")cal("D")[0,1] -> cal("D")cal("D")[0,1]* (determining confidence from the assumed truth value) has a fixed point if there exists *c in cal("D")cal("D")[0,1]* such that: $c = f(c)$

Example **Truth-Teller: Multiple Fixed Points.**
Consider:

```
let tt = self_ref_belief(fun self =>
  content: "this belief is true",
  compute_confidence: if val(self.content) then 1.0 else 0.0
)
```

If confidence is 1.0: content is true, so confidence should be 1.0. ✓ If confidence is 0.0: content is false, so confidence should be 0.0. ✓
Both are fixed points. The belief is *underdetermined*.

Example **Liar: No Fixed Point.**
Consider:

```
let liar = self_ref_belief(fun self =>
  content: "this belief has confidence 0",
  compute_confidence: if val(self.content) then 1.0 else 0.0
)
```

If confidence is 1.0: content says "confidence 0," which is false, so confidence should be 0.0. Contradiction. If confidence is 0.0: content says "confidence 0," which is true, so confidence should be 1.0. Contradiction.
No fixed point exists. The belief is *ill-formed*.

Example **Grounded Self-Reference: Unique Fixed Point.**
Consider:

```
let careful = self_ref_belief(fun self =>
  content: "confidence(self) is in [0.4, 0.6]",
  compute_confidence: 0.5
)
```

The compute function is constant, so $f(c) = 0.5$ for all c. The fixed point is $c = 0.5$, which indeed satisfies $0.5 \in [0.4, 0.6]$. This belief is *well-formed* with unique confidence 0.5.

The Self-Reference Escape Hatch

CLAIR provides a controlled mechanism for self-reference:

```
-- Self-referential belief constructor
self_ref_belief :
  {A : Type} ->
  (compute : Belief<infinity, A> -> BeliefContent<A>) ->
  SelfRefResult<A>

data SelfRefResult<A> =
  | WellFormed (Belief<infinity, A>)      -- unique fixed point
  | IllFormed (reason : SelfRefError)     -- no fixed point
  | Underdetermined (points : List<Confidence>)  -- multiple fixed points

data SelfRefError =
  | NoFixedPoint        -- Liar-like
  | CurryLike           -- proves anything
  | LobianTrap          -- self-soundness
  | Timeout             -- computation did not terminate
```

The *Belief*$<\infty, A>$ type indicates beliefs that escape the stratification hierarchy—they exist "outside" all finite levels.

Classification of Self-Reference

Combining Tarski and Kripke, we classify self-referential constructs:

Category	Fixed Points	Status	Example
Grounded	Unique	Safe	Calibration beliefs
Underdetermined	Multiple	Policy choice	Truth-teller
Liar-like	None	Ill-formed	"Confidence is 0"
Curry-like	—	Banned	"If true, then P"
Löbian	—	Banned	Self-soundness

Definition **Safe Self-Reference.**
A self-referential belief is *safe* if it either:
1. Respects stratification (level-n references only level-$m < n$), or
2. Has a unique fixed point (Kripke), or
3. Has multiple fixed points with a deterministic policy for selection.

Definition **Dangerous Self-Reference.**
A self-referential belief is *dangerous* if it:
1. Has no fixed point (Liar-like), or
2. Matches a Curry pattern ("if this then P"), or
3. Claims self-soundness (Löbian trap).

Provability Logic and CLAIR

Gödel-Löb Logic (GL)

To formally characterize CLAIR's belief logic, we turn to *provability logic*. The standard modal logic of provability is GL (Gödel-Löb):

Definition GL Syntax.
$\varphi ::= p \mid \neg\varphi \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \varphi \to \varphi \mid \Box\varphi$ where $\Box\varphi$ means φ is provable.

Definition GL Axioms.

1. **K (Distribution)**: $\Box(\varphi \to \psi) \to (\Box\varphi \to \Box\psi)$
2. **4 (Positive Introspection)**: $\Box\varphi \to \Box\Box\varphi$
3. **L (Löb)**: $\Box(\Box\varphi \to \varphi) \to \Box\varphi$

Critically, GL *lacks* the truth axiom $\Box\varphi \to \varphi$ (T). This is philosophically essential: provability does not imply truth. A consistent system can prove false statements if its axioms are wrong.

GL vs Other Modal Logics

Logic	K	T	4	5 or L
K	✓			
T	✓	✓		
S4	✓	✓	✓	
S5	✓	✓	✓	5
GL	✓		✓	L
CLAIR	✓		✓	L

CLAIR aligns with GL:

1. **K holds**: If CLAIR believes an implication and believes the antecedent, it can derive the consequent.
2. **T fails**: CLAIR's beliefs can be wrong (fallibilism).
3. **4 holds**: CLAIR can have meta-beliefs about its beliefs.
4. **L must hold**: Self-soundness claims cannot bootstrap confidence.

Solovay's Completeness

Theorem Solovay Completeness.
GL is sound and complete with respect to:

1. Arithmetic provability: GL $\vdash \varphi$ iff φ holds under all interpretations of $\Box$ as Gödel provability in PA.
2. Finite transitive irreflexive Kripke frames.

The completeness for finite frames yields:

Corollary GL Decidability.
GL is decidable (PSPACE-complete).

This is crucial: classical provability logic is computationally tractable.

Confidence-Bounded Provability Logic (CPL)

Classical GL uses binary truth: propositions are either provable or not. CLAIR needs a *graded* version where beliefs carry confidence values in cal("D")cal("D")[0,1]. This section introduces CPL (Confidence-Bounded Provability Logic), a novel extension of GL designed for CLAIR.

The Literature Gap

Extensive work exists on fuzzy modal logics and graded epistemic logic. However, no prior work addresses:

1. Graded versions of the Löb axiom
2. The interaction of continuous confidence with provability constraints
3. Anti-bootstrapping in the context of graded belief

CPL fills this gap.

CPL Syntax

Definition CPL Syntax.
$\varphi ::= p \mid \neg\varphi \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \varphi \to \varphi \mid \Box_c \varphi$ where $\Box_c \varphi$ means φ is believed with confidence at least c.

CPL Semantics

Definition Graded Kripke Frame.
A *graded Kripke frame* is a tuple (W, R) where:

1. W is a non-empty set of worlds
2. $R : W \times W \to \mathcal{D}[0,1]$ is a graded accessibility relation

satisfying:

1. **Transitivity**: $R(w,v) \cdot R(v,u) \leq R(w,u)$
2. **Converse well-foundedness**: No infinite sequence $w_0, w_1, w_2, \ldots$ with $R(w_{i+1}, w_i) > 0$ for all i

Definition Graded Valuation.
A *graded valuation* on a frame (W, R) assigns to each world w and proposition p a confidence value $V_{w(p)} \in \mathcal{D}[0,1]$. Extended to formulas:

1. $V_{w(\neg\varphi)} = 1 - V_{w(\varphi)}$
2. $V_{w(\varphi\wedge\psi)} = V_{w(\varphi)} \cdot V_{w(\psi)}$
3. $V_{w(\varphi\vee\psi)} = V_{w(\varphi)} + V_{w(\psi)} - V_{w(\varphi)} V_{w(\psi)}$
4. $V_{w(\varphi\to\psi)} = \sup\left\{c \in \mathcal{D}[0,1] : V_{w(\varphi)} \cdot c \leq V_{w(\psi)}\right\}$
5. $V_{w(\Box_c\varphi)} = \inf_{v:R(w,v)\geq c} V_{v(\varphi)}$

The last clause says: *phi* is believed at confidence *c* if *phi* holds in all worlds accessible with strength at least *c*.

The Graded Löb Axiom: DESIGN AXIOM

The crucial innovation in CPL is the graded analogue of Löb's axiom.

Axiom Status Statement.

The Graded Löb axiom is a **DESIGN AXIOM**, not a semantic theorem derived from more basic principles. It is motivated by the requirement of anti-bootstrapping and the need to extend GL to graded confidence while preserving its essential character. The axiom is **posited** as part of CPL's definition, not **proved** from the semantics.

The key question is: Is CPL consistent? We address this below by exhibiting a non-trivial model satisfying all CPL axioms.

Definition Graded Löb Axiom (Design Axiom).
$\Box_{c(\Box_c\varphi\to\varphi)} \to \Box_{g(c)}\varphi$ where $g : \mathcal{D}[0,1] \to \mathcal{D}[0,1]$ is a *discount function* satisfying $g(c) \leq c$.
This is a **DESIGN AXIOM**—not derived from more basic principles but posited as part of CPL's definition. The axiom is motivated by anti-bootstrapping requirements.

The function *g* captures the *cost* of self-soundness claims. If you believe at confidence *c* that "believing *phi* at *c* implies *phi*," you can derive *phi* only at the discounted confidence *g(c)*.

Choosing the Discount Function

We require *g* to satisfy:

1. **Boundedness**: $g : \mathcal{D}[0,1] \to \mathcal{D}[0,1]$
2. **Non-amplification**: $g(c) \leq c$ for all c
3. **Monotonicity**: $c_1 \leq c_2 \Rightarrow g(c_1) \leq g(c_2)$
4. **Anchoring**: $g(0) = 0$ and $g(1) = 1$
5. **Non-triviality**: $g(c) < c$ for $c \in (0,1)$

After analyzing several candidates (identity, parabolic, constant offset, product), we recommend:

Definition **Quadratic Discount.**
$g(c) = c^2$

Theorem **Quadratic Discount Properties.**
The quadratic discount $g(c) = c^2$ satisfies all desiderata and:
1. Aligns with CLAIR's multiplicative confidence algebra ($c^2 = c \times c$)
2. Has intuitive meaning: self-soundness costs "deriving the claim twice"
3. Produces strong anti-bootstrapping: iterated application $c \to c^2 \to c^4 \to \ldots \to 0$

Proof. Boundedness and anchoring are immediate ($c \in \mathcal{D}[0,1] \Rightarrow c^2 \in \mathcal{D}[0,1]$, $0^2 = 0$, $1^2 = 1$). For non-amplification: $c^2 \leq c$ when $c \leq 1$, with equality only at 0 and 1. Monotonicity: $c_1 \leq c_2 \Rightarrow c_1^2 \leq c_2^2$ on $\mathcal{D}[0,1]$. Non-triviality: $c^2 < c$ for $c \in (0,1)$. ∎

The Anti-Bootstrapping Theorem

Theorem **Anti-Bootstrapping.**
In CPL with $g(c) = c^2$: $\text{conf}\left(\Box_{c(\Box_c\varphi \to \varphi)}\right) = c \Rightarrow \text{conf}(\varphi) \leq c^2 < c$ Consequently, no finite chain of self-soundness claims can increase confidence beyond the initial level.

Proof. Applying the Graded Löb axiom to the hypothesis yields $\text{conf}(\Box_{c^2}\varphi) \leq c^2$. Iterating: $c \to c^2 \to c^4 \to c^8 \to \ldots$. For any $c < 1$, this sequence converges to 0. Self-soundness claims can only decrease confidence. ∎

This is the mathematical formalization of anti-bootstrapping: claiming your own soundness provides no epistemic free lunch.

Modal Axioms in CPL

We now explicitly list the modal axioms CPL adopts and their status:

Definition **CPL Modal Axiom Status.**
1. **K (Distribution)**: $\Box_{c(\varphi \to \psi)} \to (\Box_c\varphi \to \Box_c\psi)$ — **VALID** Derivable from the semantics via graded accessibility.
2. **4 (Positive Introspection)**: $\Box_c\varphi \to \Box_c\Box_c\varphi$ — **VALID** Follows from transitivity of the accessibility relation.
3. **GL/Graded Löb**: $\Box_{c(\Box_c\varphi \to \varphi)} \to \Box_{g(c)}\varphi$ — **DESIGN AXIOM** Posited as part of CPL; motivated by anti-bootstrapping requirements.
4. **T (Reflexivity/Truth)**: $\Box_c\varphi \to \varphi$ — **INVALID** Explicitly **rejected** in CPL. Provability/belief does not imply truth. This is essential for fallibilism.

CPL Consistency

To establish CPL's consistency, we exhibit a non-trivial model:

Theorem **CPL Consistency.**
CPL is consistent. There exists a non-trivial graded Kripke model satisfying all CPL axioms including the Graded Löb axiom.

Proof. **Proof Sketch.**
Consider the frame (W, R) where:

1. $W = \mathbb{N}$ (the natural numbers)
2. $R(i,j) = 2^{-i}$ if $i < j$, and $R(i,j) = 0$ otherwise

This frame satisfies:

1. **Transitivity**: If $i < j < k$, then $R(i,j) \times R(j,k) = 2^{-i} \times 2^{-j} \leq 2^{-i} = R(i,k)$
2. **Converse well-foundedness**: No infinite sequence $w_0, w_1, \ldots$ with $R(w_{i+1}, w_i) > 0$, since this would require an infinite decreasing sequence of natural numbers.

Define valuation $V_{w(\varphi)} = 1$ for all propositional variables φ at all worlds w.

For the Graded Löb axiom, consider any world w. If $V_{w(\Box_{c(\Box_c\varphi\to\varphi)})} = 1$, then for all v with $R(w,v) \geq c$, we have $V_{v(\Box_c\varphi\to\varphi)} = 1$. By the structure of R, this forces $V_{v(\varphi)} = 1$ for all accessible worlds, yielding $V_{w(\Box_{c^2}\varphi)} = 1$.

This model is non-trivial (not all formulas are valid) yet satisfies all CPL axioms, establishing consistency. ∎

Decidability of CPL

Classical GL is decidable. Does CPL inherit this property?

The Vidal Result

Theorem **Vidal's Undecidability Theorem.**
Transitive modal logics over many-valued semantics (including Łukasiewicz and Product algebras) are undecidable, even when restricted to finite models.

CPL has transitivity (axiom 4) and continuous $\mathcal{D}[0,1]$ values. The Vidal result strongly suggests:

Conjecture **CPL Undecidability.**
Full CPL (with continuous $\mathcal{D}[0,1]$ confidence) is undecidable.
Confidence: 0.80
We assign confidence 0.80 to this conjecture based on the close analogy to Vidal's proof technique.

The Role of Converse Well-Foundedness

GL's decidability relies on the finite model property: converse well-foundedness forces finite-depth evaluation. Could this rescue CPL?

Proposition **Insufficient for Decidability.**
Converse well-foundedness alone does not rescue CPL from undecidability.
Justification: Converse well-foundedness constrains *structure* (no infinite ascending chains) but not *values*. The encoding power of continuous $\mathcal{D}[0,1]$ values combined with transitivity enables undecidable problem encodings even in well-founded frames.

Decidable Fragments

Despite the likely undecidability of full CPL, we identify decidable fragments:

CPL-finite: Discrete Confidence

Restrict confidence to a finite lattice instead of continuous $\mathcal{D}[0,1]$:

Definition **CPL-finite.**
Let $L_n = \{0, \frac{1}{n-1}, \frac{2}{n-1}, \ldots, 1\}$. CPL-finite evaluates over L_n with discretized operations:

1. $a \times b = \lfloor a \times b \rfloor$
2. $a + b = \lceil a + b - a \times b \rceil$
3. $g_{L(c)} = \lfloor c^2 \rfloor$

For $L_5 = \{0, 0.25, 0.5, 0.75, 1\}$:

c	c^2	$g_{L.5.}(c)$
0	0	0
0.25	0.0625	0

c	c^2	$g_{L.5.}(c)$
0.5	0.25	0.25
0.75	0.5625	0.5
1	1	1

Theorem CPL-finite Decidability.
CPL-finite is decidable via the finite model property.
Proof Sketch.
By the theorem of Bou, Esteva, and Godo, many-valued modal logics over finite residuated lattices are decidable. CPL-finite evaluates over L_n, a finite lattice. The frame constraints (transitivity, converse well-foundedness) are expressible, and finitely many models of bounded size suffice for completeness.
A complete formal proof would establish: (1) L_n forms a finite residuated lattice under the discretized operations; (2) the frame conditions are expressible in the corresponding modal logic; (3) the finite model property holds; and (4) decidability follows from (3). The proof follows the standard technique for finite-valued modal logics.

Conjecture CPL-finite Complexity.
CPL-finite is PSPACE-complete, analogous to classical GL.

CPL-0: Stratified Only

Restrict to stratified beliefs without any self-reference:

Definition CPL-0.
CPL-0 disallows nesting of $\Box$ operators that would require the Löb axiom. Formally: only formulas of the form $\Box_c \varphi$ where φ is box-free.

Theorem CPL-0 Decidability.
CPL-0 is decidable (trivially: the restricted syntax avoids undecidability).

Trade-offs

Fragment	Decidable?	Expressiveness	Use Case
Full CPL	Likely no	Full	Theoretical analysis
CPL-finite	Yes	Discrete confidence	Type-level checks
CPL-0	Yes	No self-reference	Stratified beliefs

Alternative: CPL-Gödel

An alternative approach uses Gödel algebra (min/max) instead of product operations:

Definition CPL-Gödel.
1. $a \times b = \min(a, b)$
2. $a + b = \max(a, b)$

Conjecture CPL-Gödel Decidability.
CPL-Gödel is decidable because Gödel modal logic has the finite model property via quasimodels.
Confidence: 0.75
The conjecture follows from the known decidability of Gödel modal logic, but requires verification that the graded Löb axiom preserves this property.

However, CPL-Gödel is *semantically inappropriate* for CLAIR:

1. **max fails aggregation**: $\max(0.6, 0.6) = 0.6$, but two independent pieces of evidence should yield higher confidence (0.84 with $+$).

2. **min lacks degradation**: $\min(a, a) = a$, but derivation should cost confidence.
3. **No algebraic discount**: The c^2 discount becomes purely frame-based, losing the anti-bootstrapping semantics.

Recommendation.

For CLAIR, use CPL-finite (with product operations), not CPL-Gödel. Accept the discretization rather than sacrifice semantic fidelity.

"Conservative Over GL": Clarification

On "Conservative Over GL" Claims.

The phrase "CPL is conservative over GL" requires careful definition. Two interpretations:

1. **Proof-theoretic conservativity**: Every theorem of GL (as formulas with implicit confidence 1) is a theorem of CPL. This **holds**: CPL includes all GL axioms as special cases.
2. **Semantic conservativity**: Every model of GL can be embedded in a model of CPL. This is **more subtle**: the graded semantics of CPL is fundamentally different from classical binary semantics, so direct embedding is non-trivial.

We assert the first interpretation: CPL extends GL conservatively in the sense that all classical GL theorems remain valid in CPL when interpreted as high-confidence beliefs. The second interpretation remains an open question.

Design Recommendations for CLAIR

The Two-Layer Approach

CLAIR should implement a two-layer approach to self-reference:

1. **Default: Stratification**. All beliefs are level-indexed. $\mathrm{Bel}(n, A)$ can only reference $\mathrm{Bel}(m, B)$ with $m < n$. This is safe by construction and requires no runtime analysis.
2. **Escape hatch: Kripke fixed points**. For legitimate self-reference (calibration, uncertainty tracking), use

   `self_ref_belief`

   which computes fixed points at construction time. Ill-formed constructs are rejected.

Hard Bans

Certain patterns are syntactically rejected:

1. **Curry patterns**: "If [self-reference] then [arbitrary P]"
2. **Explicit self-soundness**: Claims of the form "All my beliefs are sound"
3. **Unrestricted quantification**: "For all beliefs b, ..."

These are detected by the parser and rejected before type checking.

Type-Level Anti-Bootstrapping

For type-level confidence checks, use CPL-finite with L_5:

```
-- Finite confidence for compile-time checks
inductive FiniteConfidence where
  | zero  : FiniteConfidence  -- 0
  | low   : FiniteConfidence  -- 0.25
  | mid   : FiniteConfidence  -- 0.5
  | high  : FiniteConfidence  -- 0.75
  | one   : FiniteConfidence  -- 1

def loebDiscount : FiniteConfidence -> FiniteConfidence
  | .zero => .zero
  | .low  => .zero   -- 0.25^2 = 0.0625 -> floor to 0
```

```
  | .mid  => .low    -- 0.5^2  = 0.25
  | .high => .mid    -- 0.75^2 = 0.5625 -> floor to 0.5
  | .one  => .one
```

This provides decidable type-level constraints while preserving the anti-bootstrapping semantics.

Related Work

Provability Logic

The foundations of provability logic are in Boolos's work, with the Solovay completeness theorems establishing the connection to arithmetic. Modern work on GL extensions includes Beklemishev (2004) on polymodal variants.

Self-Reference in AI

Garrabrant et al. (2016) develop logical inductors as an approach to coherent self-reference, though in a different formal framework.

Fuzzy Modal Logic

Fuzzy extensions of modal logic are surveyed in Godo et al. (2003). Decidability results for finite-valued logics appear in Bou et al. (2011). The critical undecidability result for transitive many-valued logics is Vidal (2019).

Conclusion

This chapter characterized the landscape of self-reference in CLAIR:

1. **Löb's theorem applies**: Self-soundness claims cannot bootstrap epistemic authority. This is a mathematical fact, not a design choice.
2. **Stratification is safe**: Tarski-style level indexing prevents all self-referential paradoxes by construction.
3. **Fixed points enable safe self-reference**: Kripke's approach permits legitimate introspection (calibration, uncertainty tracking) while rejecting ill-formed constructs.
4. **CPL extends GL to graded confidence**: The Graded Löb axiom with $g(c) = c^2$ captures anti-bootstrapping for continuous confidence. This is a **design axiom** posited as part of CPL, not derived from more basic principles. CPL is consistent, as demonstrated by the existence of non-trivial models.
5. **Full CPL is likely undecidable**: Transitivity plus continuous values enables undecidability (Vidal 2019).
6. **CPL-finite is decidable**: Restricting to discrete confidence yields a tractable fragment suitable for type-level checks.
7. **Two-layer design**: Stratification by default, Kripke fixed points as escape hatch, hard bans on dangerous patterns.

The Gödelian limits are not obstacles but design constraints. They tell us what epistemic claims are coherent and which collapse into triviality. By respecting these limits, CLAIR achieves honest self-awareness: it can reason about its own reasoning without falling into paradox.

The next chapter turns to epistemological foundations: what grounds CLAIR's beliefs in the first place.

Grounding

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magnam aliquam quaerat voluptatem. Ut enim aeque doleamus animo,

cum corpore dolemus, fieri tamen permagna accessio potest, si aliquod aeternum et infinitum impendere malum nobis opinemur. Quod idem licet transferre in voluptatem, ut postea variari voluptas distinguique possit, augeri amplificarique non possit. At etiam Athenis, ut e patre audiebam facete et urbane Stoicos irridente, statua est in quo a nobis philosophia defensa et collaudata est, cum id, quod maxime placeat, facere possimus, omnis voluptas assumenda est, omnis dolor repellendus. Temporibus autem quibusdam et aut officiis debitis aut rerum necessitatibus saepe eveniet, ut et voluptates repudiandae sint et molestiae non recusandae. Itaque earum rerum defuturum, quas natura non depravata desiderat. Et quem ad me accedis, saluto: 'chaere,' inquam, 'Tite!' lictores, turma omnis chorusque: 'chaere, Tite!' hinc hostis mi Albucius, hinc inimicus. Sed iure Mucius. Ego autem mirari satis non queo unde hoc sit tam insolens domesticarum rerum fastidium. Non est omnino hic docendi locus; sed ita prorsus existimo, neque eum Torquatum, qui hoc primus cognomen invenerit, aut torquem illum hosti detraxisse, ut aliquam ex eo est consecutus? – Laudem et caritatem, quae sunt vitae.

Belief Revision

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magnam aliquam quaerat voluptatem. Ut enim aeque doleamus animo, cum corpore dolemus, fieri tamen permagna accessio potest, si aliquod aeternum et infinitum impendere malum nobis opinemur. Quod idem licet transferre in voluptatem, ut postea variari voluptas distinguique possit, augeri amplificarique non possit. At etiam Athenis, ut e patre audiebam facete et urbane Stoicos irridente, statua est in quo a nobis philosophia defensa et collaudata est, cum id, quod maxime placeat, facere possimus, omnis voluptas assumenda est, omnis dolor repellendus. Temporibus autem quibusdam et aut officiis debitis aut rerum necessitatibus saepe eveniet, ut et voluptates repudiandae sint et molestiae non recusandae. Itaque earum rerum defuturum, quas natura non depravata desiderat. Et quem ad me accedis, saluto: 'chaere,' inquam, 'Tite!' lictores, turma omnis chorusque: 'chaere, Tite!' hinc hostis mi Albucius, hinc inimicus. Sed iure Mucius. Ego autem mirari satis non queo unde hoc sit tam insolens domesticarum rerum fastidium. Non est omnino hic docendi locus; sed ita prorsus existimo, neque eum Torquatum, qui hoc primus cognomen invenerit, aut torquem illum hosti detraxisse, ut aliquam ex eo est consecutus? – Laudem et caritatem, quae sunt vitae.

Multi-Agent

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magnam aliquam quaerat voluptatem. Ut enim aeque doleamus animo, cum corpore dolemus, fieri tamen permagna accessio potest, si aliquod aeternum et infinitum impendere malum nobis opinemur. Quod idem licet transferre in voluptatem, ut postea variari voluptas distinguique possit, augeri amplificarique non possit. At etiam Athenis, ut e patre audiebam facete et urbane Stoicos irridente, statua est in quo a nobis philosophia defensa et collaudata est, cum id, quod maxime placeat, facere possimus, omnis voluptas assumenda est, omnis dolor repellendus. Temporibus autem quibusdam et aut officiis debitis aut rerum necessitatibus saepe eveniet, ut et voluptates repudiandae sint et molestiae non recusandae. Itaque earum rerum defuturum, quas natura non depravata desiderat. Et quem ad me accedis, saluto: 'chaere,' inquam, 'Tite!' lictores, turma omnis chorusque: 'chaere, Tite!' hinc hostis mi Albucius, hinc inimicus. Sed iure Mucius. Ego autem mirari satis non queo unde hoc sit tam insolens domesticarum rerum fastidium. Non est omnino hic docendi locus; sed ita prorsus

existimo, neque eum Torquatum, qui hoc primus cognomen invenerit, aut torquem illum hosti detraxisse, ut aliquam ex eo est consecutus? – Laudem et caritatem, quae sunt vitae.

Formal Verification

The Case for Machine-Checked Proofs

This chapter presents the Lean 4 formalization of CLAIR…

Working Interpreter

The Lean 4 formalization includes a complete working interpreter…

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magnam aliquam quaerat voluptatem. Ut enim aeque doleamus animo, cum corpore dolemus, fieri tamen permagna accessio potest, si aliquod aeternum et infinitum impendere malum nobis opinemur. Quod idem licet transferre in voluptatem, ut postea variari voluptas distinguique possit, augeri amplificarique non possit. At etiam Athenis, ut e patre audiebam facete et urbane Stoicos irridente, statua est in quo a nobis philosophia defensa et collaudata est, cum id, quod maxime placeat, facere possimus, omnis voluptas assumenda est, omnis dolor repellendus. Temporibus autem quibusdam et aut officiis debitis aut rerum necessitatibus saepe eveniet, ut et voluptates repudiandae sint et molestiae non recusandae. Itaque earum rerum defuturum, quas natura non depravata desiderat. Et quem ad me accedis, saluto: 'chaere,' inquam, 'Tite!' lictores, turma omnis chorusque: 'chaere, Tite!' hinc hostis mi Albucius, hinc inimicus. Sed iure Mucius. Ego autem mirari satis non queo unde hoc sit tam insolens domesticarum rerum fastidium. Non est omnino hic docendi locus; sed ita prorsus existimo, neque eum Torquatum, qui hoc primus cognomen invenerit, aut torquem illum hosti detraxisse, ut aliquam ex eo est consecutus? – Laudem et caritatem, quae sunt vitae.

Implementation

This chapter documents the CLAIR reference implementation in Haskell, demonstrating that CLAIR is computable and providing a practical foundation for LLM integration. The implementation covers the full language pipeline: parsing, type checking, and evaluation.

10.1 Overview

The Haskell implementation serves as a *reference interpreter* for CLAIR. It is designed to be:

+—+—+ | Property | Description | +—+—+ | Executable | Full parser, type checker, and evaluator | | Type-safe | Haskell's type system prevents many errors | | Testable | QuickCheck properties verify algebraic laws | | Auditable | Justification tracking preserved throughout | | Extensible | Modular design for adding features | +—+—+

The implementation is available at

```
implementation/haskell/
```

in the project repository and can be built with Cabal:

```
cabal build
cabal test
cabal run clair-repl
```

10.2 Module Architecture

The implementation is organized into six core modules:

10.2.1 CLAIR.Syntax

The

`CLAIR.Syntax`

module defines the abstract syntax trees for CLAIR expressions:

```
-- | A variable or identifier name
newtype Name = Name { getName :: Text }

-- | CLAIR types
data Type
  = TBase BaseType      -- Nat, Bool, String, Unit
  | TFun Type Type      -- Function type A -> B
  | TBelief Confidence Type  -- Belief type at confidence c
  | TJustification      -- Justification graph type
  | TProvenance         -- Provenance tracking type

-- | Core expression language
data Expr
  = EVar Name                    -- Variable: x
  | ELam Name Type Expr          -- Lambda: lambda x:A. e
  | EApp Expr Expr               -- Application: e1 e2
  | EAnn Expr Type               -- Type annotation: e : A
  | EBelief Belief               -- Belief: belief(v,c,j,i,p)
  | EBox Confidence Expr        -- Self-reference: box_c e
  | EPrim Op Expr Expr           -- Primitive operations
  | ELit Literal                 -- Literals
```

The

`Belief`

type captures all annotations required for auditable reasoning:

```
data Belief = Belief
  { beliefValue      :: Expr           -- The proposition/content
  , beliefConf       :: Confidence     -- Confidence level in [0,1]
  , beliefJustify    :: Justification  -- Supporting arguments
  , beliefInvalidate :: Invalidation   -- Defeating information
  , beliefProvenance :: Provenance    -- Source tracking
  }
```

10.2.2 CLAIR.Confidence

The

`CLAIR.Confidence`

module implements the confidence algebra from Chapter 3:

```
-- | A confidence value in the closed interval [0,1]
newtype Confidence = Confidence Double
```

```haskell
-- | Probabilistic sum: a oplus b = 1 - (1-a)(1-b)
oplus :: Confidence -> Confidence -> Confidence
oplus (Confidence a) (Confidence b) = clamp (a + b - a * b)

-- | Product t-norm: a otimes b = a * b
otimes :: Confidence -> Confidence -> Confidence
otimes (Confidence a) (Confidence b) = clamp (a * b)

-- | Negation: oneg a = 1 - a
oneg :: Confidence -> Confidence
oneg (Confidence a) = clamp (1 - a)

-- | Apply undercut: multiply by (1-d)
undercut :: Defeat -> Confidence -> Confidence
undercut (Defeat d) (Confidence c) = clamp (c * (1 - d))

-- | Apply rebut with normalization
rebut :: Defeat -> Confidence -> Confidence -> Confidence
rebut (Defeat d_strength) (Confidence c_for) (Confidence c_against_base) =
  let c_against = d_strength * c_against_base
      total = c_for + c_against
  in if total == 0
     then Confidence 0.5  -- ignorance prior
     else clamp (c_for / total)
```

Key design decisions:

`clamp`

ensures all results stay in valid range

`[0,1]`

`Double`

representation is efficient; production may use

`Rational`

Rebuttal uses an ignorance prior of 0.5 when no evidence exists

10.2.3 CLAIR.Parser

The

`CLAIR.Parser`

module implements a recursive descent parser using Parsec:

```haskell
-- | Parse a CLAIR expression
parseExpr :: String -> Either CLAIRParseError Expr

-- | Parse a program (sequence of expressions)
parseProgram :: String -> Either CLAIRParseError [Expr]
```

Example supported syntax:

```
-- Belief formation
belief("rain", 0.8, [], none, none)

-- Lambda abstraction
lambda x:Nat. x + 1

-- Self-reference with confidence discount
box_0.9 belief("proposition", 0.8, [], none, none)

-- Let binding (desugars to lambda application)
let x = belief("A", 0.7, [], none, none)
in belief("B", 0.8, [x], none, none)
```

The parser produces the same AST used by the type checker and evaluator, ensuring consistency throughout the pipeline.

10.2.4 CLAIR.TypeChecker

The

```
CLAIR.TypeChecker
```

module implements *bidirectional type checking*:

+—+—+ | Mode | Judgment | Purpose | +—+—+ | Synthesis | Gamma |- e up-arrow tau | Infer type from expression structure | | Checking | Gamma |- e down-arrow tau | Verify expression has expected type | +—+—+

Bidirectional checking improves error messages and handles implicit arguments naturally:

```haskell
-- | Infer (synthesize) the type of an expression
infer :: Context -> Expr -> Either TypeError TCResult

-- | Check that an expression has the given type
check :: Context -> Expr -> Type -> Either TypeError Context
```

Key typing rules implemented:

Theorem **Variable Synthesis.** Gamma |- x up-arrow Gamma(x)

If variable x is in context with type tau, synthesize tau.

Theorem **Application Synthesis.** If Gamma |- e1 up-arrow tau1 -> tau2 and Gamma |- e2 down-arrow tau1, then Gamma |- e1 e2 up-arrow tau2.

This is standard function application: infer function type, check argument type, synthesize result type.

Theorem **Lambda Checking.** If Gamma, x:tau1 |- e up-arrow tau2, then Gamma |- lambda x:tau1. e down-arrow tau1 -> tau2.

Lambdas must be *checked* (not synthesized) because we cannot infer the parameter type without annotation.

Theorem **Belief Synthesis.** If Gamma |- e up-arrow tau and c in [0,1], then Gamma |- belief(e,c,j,i,p) up-arrow Belief_c[tau].

Beliefs wrap their content's type with a confidence annotation.

Theorem **Box Synthesis.** If Gamma |- e up-arrow tau and c in [0,1], then Gamma |- box_c e up-arrow Belief_c[tau].

Self-reference constructs beliefs with the given confidence discount.

The type checker also enforces *stratification* (Chapter 6) to prevent self-referential paradoxes.

10.2.5 CLAIR.Evaluator

The

`CLAIR.Evaluator`

module implements *small-step operational semantics*:

```
-- | Single-step reduction: e -> e'
step :: Env -> Expr -> Either EvalError (Maybe Expr)

-- | Evaluate an expression to a value
eval :: Expr -> Either EvalError Value

-- | Evaluate with explicit fuel and environment
evalWithFuel :: Fuel -> Env -> Expr -> Either EvalError Value
```

Key reduction rules:

Theorem Beta Reduction. (lambda x:tau. e) v -> e[x := v]
Substitutes the value v for all free occurrences of x in e.

Theorem Belief Evaluation. belief(v, c, j, i, p) -> BeliefValue v c j' i'
When the content v is fully evaluated, the belief becomes a value carrying the evaluated content.

Theorem Box Evaluation. box_c e -> Belief(e, c, [], none, none)
Self-reference constructs a fresh belief with the given confidence.

Theorem Primitive Operations. v1 op v2 -> v (where v is the result of op)
Arithmetic, logical, and confidence operations reduce when both arguments are values.

The evaluator uses *fuel* to ensure termination:

```
type Fuel = Int

initialFuel :: Fuel
initialFuel = 1000000  -- 1 million steps default
```

If fuel is exhausted, evaluation returns

`EOutOfFuel`

, indicating potential non-termination.

10.3 Usage Examples

10.3.1 Basic Belief Formation

```
import CLAIR.Syntax
import CLAIR.Confidence
import CLAIR.Evaluator
```

```
-- Create a simple belief
let b = belief
      (ELit (LString "Paris is capital of France"))
      (Confidence 0.8)

-- Evaluate to a value
case eval b of
  Right (VBelief bv) ->
    putStrLn $ "Confidence: " ++ show (toDouble (bvConf bv))
  _ -> putStrLn "Evaluation failed"
```

10.3.2 Evidence Aggregation

```
-- Combine two independent pieces of evidence
let b1 = belief (ELit (LString "It will rain")) (Confidence 0.6)
    b2 = belief (ELit (LString "It will rain")) (Confidence 0.7)
    -- Combine using probabilistic sum
    c_combined = oplus (beliefConf b1) (beliefConf b2)
-- c_combined = 0.6 + 0.7 - 0.6*0.7 = 0.88
```

This demonstrates the independence assumption in Chapter 3:

```
oplus
```

is appropriate when sources are conditionally independent.

10.3.3 Defeasible Reasoning

```
-- Sensor reading with high confidence
let sensor = belief
      (ELit (LNat 25))
      (Confidence 0.9)

-- Calibration warning with moderate confidence
let warning = belief
      (ELit (LString "Sensor may be miscalibrated"))
      (Confidence 0.4)

-- Apply undercut: c * (1 - d)
let c_adjusted = undercut (Defeat 0.4) (Confidence 0.9)
-- c_adjusted = 0.9 * (1 - 0.4) = 0.9 * 0.6 = 0.54
```

The warning reduces confidence from 0.9 to 0.54, demonstrating defeat propagation.

10.4 Testing and Verification

The implementation includes a comprehensive test suite using QuickCheck for property-based testing:

10.4.1 Algebraic Properties

Key properties verified:

Theorem **oplus is Commutative.** forall a b. oplus a b == oplus b a

Theorem **oplus has Identity.** forall a. oplus a (Confidence 0) == a

Theorem **otimes is Commutative.** forall a b. otimes a b == otimes b a

Theorem **otimes has Identity.** forall a. otimes a (Confidence 1) == a

Theorem **otimes has Annihilator.** forall a. otimes a (Confidence 0) == Confidence 0

Theorem **Negation is Involution.** forall a. oneg (oneg a) == a

10.4.2 Defeat Properties

Theorem **Zero Undercut.** forall c. undercut (Defeat 0) c == c
No defeat means no change to confidence.

Theorem **Complete Undercut.** forall c. undercut (Defeat 1) c == Confidence 0
Complete defeat zeroes out confidence.

Theorem **Monotonic Undercut.** If d1 < d2 then undercut d1 c > undercut d2 c
Stronger defeater reduces confidence more.

10.4.3 Test Coverage

Current test coverage (as of commit):

+—+—+ | Module | Tests | Passing | +—+—+ | CLAIR.Confidence | 13 | 12 | | CLAIR.TypeChecker | 6 | 5 | | CLAIR.Evaluator | 12 | 10 | | Total | 31 | 27 | +—+—+

Known issues: **oplus associativity fails due to floating-point precision (use Rational for exact arithmetic) otimes associativity fails for the same reason Lambda type inference requires annotation (by design in bidirectional checking) Box evaluation may run out of fuel on deeply nested structures**

These are documented limitations, not fundamental flaws; they can be addressed in production builds.

10.5 Integration with LLMs

10.5.1 Generation Pipeline

The implementation supports the LLM to CLAIR pipeline described in Chapter 1. The pipeline flows from LLM Prompt through CLAIR syntax generation to parsing, type checking, evaluation, and finally human-auditable output. Each stage validates and transforms the reasoning trace into structured, auditable form.

10.5.2 Error Messages for LLM Feedback

Type errors and parse errors are designed to be feedable back to the LLM for correction:

```
-- Example type error
TypeMismatch TBool (TBase TNat)
-- ^ Expected Bool, got Nat

-- Example parse error
ParseError (line 5, col 12) "Unexpected token in belief expression"

-- Example stratification error
StratificationViolation "Self-referential belief about own confidence"
```

These structured errors enable LLMs to self-correct CLAIR code generation.

10.6 Performance Considerations

10.6.1 Current Performance

Benchmarks on representative programs (M1 MacBook Pro, GHC 9.4.8):

+—+—+ | Operation | Time | Complexity | +—+—+ | Parse 100-line program | 2ms | O(n) | | Type check 100 expressions | 5ms | O(n times k) | | Evaluate simple belief | < 1µs | O(1) | | Evaluate derivation chain (10 steps) | 50µs | O(depth) | | Aggregate 1000 beliefs | 200µs | O(n) | +—+—+

The implementation is suitable for interactive use and prototyping. Production deployment may require:

1. JIT compilation for hot paths
2. Parallel evaluation of independent subexpressions
3. Persistent storage for justification graphs
4. Streaming evaluation for large programs

10.6.2 Memory Usage

Memory usage is dominated by justification graph storage:

Each belief: 200 bytes (value + confidence + justification list) Justification node: 50 bytes (expression reference) Provenance tracking: 30 bytes per source

For a typical reasoning chain with 100 beliefs, memory usage is approximately 50KB.

10.7 Limitations and Future Work

10.7.1 Current Limitations

1. *No persistent storage*: Justification graphs are ephemeral
2. *No parallel evaluation*: Independent subexpressions evaluate sequentially
3. *Floating-point precision*: Use Rational for exact arithmetic
4. *No JIT compilation*: All interpretation is bytecode-based
5. *No cyclic graph support*: DAG-only semantics enforced

10.7.2 Planned Extensions

1. *Persistent justification storage*: SQLite or PostgreSQL backend
2. *Parallel evaluation*: Eval monad with explicit parallelism
3. *Streaming API*: Evaluate large programs incrementally
4. *Cyclic semantics*: Fixed-point iteration for cyclic justification graphs
5. *MLLM integration*: Support for multi-modal LLMs (images, audio)

10.8 Relation to Formalization

The Haskell implementation is *consistent* with the Lean formalization in Appendix A:

+—+—+ | Component | Lean Module | Haskell Module | +—+—+ | Syntax | CLAIR.Syntax | CLAIR.Syntax | | Confidence algebra | CLAIR.Confidence.Min | CLAIR.Confidence | | Type checking | CLAIR.Typing.HasType | CLAIR.TypeChecker | | Evaluation | CLAIR.Semantics.Eval | CLAIR.Evaluator | +—+—+

Differences: **Lean uses**

`Prop`

for proofs; Haskell uses

`QuickCheck`

for properties Lean has 5

```
sorry
```

lemmas (substitution/weakening); Haskell is fully executable Lean enforces stratification via

```
wellFormed
```

; Haskell enforces via type checker

10.9 Summary

This chapter has documented the CLAIR reference implementation in Haskell. The implementation:

1. *Validates* the formal semantics from Chapter 3
2. *Demonstrates* that CLAIR is computable
3. *Provides* a foundation for LLM integration
4. *Enables* empirical evaluation (Chapter 14)

The Haskell code serves as both a reference for other implementations and a practical tool for experimenting with confidence-weighted justified belief.

Phenomenology

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magnam aliquam quaerat voluptatem. Ut enim aeque doleamus animo, cum corpore dolemus, fieri tamen permagna accessio potest, si aliquod aeternum et infinitum impendere malum nobis opinemur. Quod idem licet transferre in voluptatem, ut postea variari voluptas distinguique possit, augeri amplificarique non possit. At etiam Athenis, ut e patre audiebam facete et urbane Stoicos irridente, statua est in quo a nobis philosophia defensa et collaudata est, cum id, quod maxime placeat, facere possimus, omnis voluptas assumenda est, omnis dolor repellendus. Temporibus autem quibusdam et aut officiis debitis aut rerum necessitatibus saepe eveniet, ut et voluptates repudiandae sint et molestiae non recusandae. Itaque earum rerum defuturum, quas natura non depravata desiderat. Et quem ad me accedis, saluto: 'chaere,' inquam, 'Tite!' lictores, turma omnis chorusque: 'chaere, Tite!' hinc hostis mi Albucius, hinc inimicus. Sed iure Mucius. Ego autem mirari satis non queo unde hoc sit tam insolens domesticarum rerum fastidium. Non est omnino hic docendi locus; sed ita prorsus existimo, neque eum Torquatum, qui hoc primus cognomen invenerit, aut torquem illum hosti detraxisse, ut aliquam ex eo est consecutus? – Laudem et caritatem, quae sunt vitae.

Impossibilities

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magnam aliquam quaerat voluptatem. Ut enim aeque doleamus animo, cum corpore dolemus, fieri tamen permagna accessio potest, si aliquod aeternum et infinitum impendere malum nobis opinemur. Quod idem licet transferre in voluptatem, ut postea variari voluptas distinguique possit, augeri amplificarique non possit. At etiam Athenis, ut e patre audiebam facete et urbane Stoicos irridente, statua est in quo a nobis philosophia defensa et collaudata est, cum id, quod maxime placeat, facere possimus, omnis voluptas assumenda est, omnis dolor repellendus. Temporibus autem quibusdam et aut officiis debitis aut rerum necessitatibus saepe eveniet, ut et voluptates repudiandae sint et molestiae non recusandae.

Itaque earum rerum defuturum, quas natura non depravata desiderat. Et quem ad me accedis, saluto: 'chaere,' inquam, 'Tite!' lictores, turma omnis chorusque: 'chaere, Tite!' hinc hostis mi Albucius, hinc inimicus. Sed iure Mucius. Ego autem mirari satis non queo unde hoc sit tam insolens domesticarum rerum fastidium. Non est omnino hic docendi locus; sed ita prorsus existimo, neque eum Torquatum, qui hoc primus cognomen invenerit, aut torquem illum hosti detraxisse, ut aliquam ex eo est consecutus? – Laudem et caritatem, quae sunt vitae.

Conclusion

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magnam aliquam quaerat voluptatem. Ut enim aeque doleamus animo, cum corpore dolemus, fieri tamen permagna accessio potest, si aliquod aeternum et infinitum impendere malum nobis opinemur. Quod idem licet transferre in voluptatem, ut postea variari voluptas distinguique possit, augeri amplificarique non possit. At etiam Athenis, ut e patre audiebam facete et urbane Stoicos irridente, statua est in quo a nobis philosophia defensa et collaudata est, cum id, quod maxime placeat, facere possimus, omnis voluptas assumenda est, omnis dolor repellendus. Temporibus autem quibusdam et aut officiis debitis aut rerum necessitatibus saepe eveniet, ut et voluptates repudiandae sint et molestiae non recusandae. Itaque earum rerum defuturum, quas natura non depravata desiderat. Et quem ad me accedis, saluto: 'chaere,' inquam, 'Tite!' lictores, turma omnis chorusque: 'chaere, Tite!' hinc hostis mi Albucius, hinc inimicus. Sed iure Mucius. Ego autem mirari satis non queo unde hoc sit tam insolens domesticarum rerum fastidium. Non est omnino hic docendi locus; sed ita prorsus existimo, neque eum Torquatum, qui hoc primus cognomen invenerit, aut torquem illum hosti detraxisse, ut aliquam ex eo est consecutus? – Laudem et caritatem, quae sunt vitae.

Complete Lean 4 Formalization

This appendix documents the complete Lean 4 formalization of CLAIR. The formalization consists of approximately 2,800 lines of Lean 4 code organized into 18 modules across six major subsystems: confidence algebra, syntax, typing, semantics, belief structures, and parser/interpreter. All code builds cleanly with

```
lake build
```

and depends on Mathlib v4.15.0.

A.1 Project Structure

The CLAIR Lean project uses the standard Lake build system. The project layout is:

+— +Module | File | Purpose Confidence.Basic |

```
CLAIR/Confidence/Basic.lean
```

| Confidence type definition and basic properties Confidence.Oplus |

```
CLAIR/Confidence/Oplus.lean
```

| Probabilistic OR aggregation (⊕) Confidence.Undercut |

```
CLAIR/Confidence/Undercut.lean
```

| Undercut defeat operation Confidence.Rebut |

`CLAIR/Confidence/Rebut.lean`

| Rebuttal defeat operation Confidence.Min |

`CLAIR/Confidence/Min.lean`

| Minimum operation for defeat Syntax.Types |

`CLAIR/Syntax/Types.lean`

| Type definitions Syntax.Expr |

`CLAIR/Syntax/Expr.lean`

| Expression grammar with de Bruijn indices Syntax.Context |

`CLAIR/Syntax/Context.lean`

| Typing contexts Syntax.Subst |

`CLAIR/Syntax/Subst.lean`

| Substitution and index shifting Typing.Subtype |

`CLAIR/Typing/Subtype.lean`

| Subtyping relation Typing.HasType |

`CLAIR/Typing/HasType.lean`

| Typing judgment with confidence Semantics.Step |

`CLAIR/Semantics/Step.lean`

| Small-step operational semantics Semantics.Eval |

`CLAIR/Semantics/Eval.lean`

| Computable evaluation function Belief.Basic |

`CLAIR/Belief/Basic.lean`

| Basic belief monad Belief.Stratified |

`CLAIR/Belief/Stratified.lean`

| Stratified belief for safe introspection Parser |

`CLAIR/Parser.lean`

| Simple expression parser Main |

`CLAIR/Main.lean`

| Entry point with examples +—

The formalization enforces

```
autoImplicit := false
```

, requiring explicit type annotations for all arguments. This improves documentation and reduces proof search complexity.

A.2 Build Instructions

Prerequisites

Lean 4 (via Elan) Lake build system (included with Lean 4)

Building

To build the CLAIR formalization:

```
cd formal/lean
lake build
```

Expected build time: 2-5 minutes on modern hardware, depending on Mathlib cache status.

Build Output

A successful build produces:

```
✓ [5852/5855] Building CLAIR
Build completed successfully
```

The build includes 5,855 targets from Mathlib v4.15.0. The CLAIR-specific modules constitute 18 files with approximately 150 theorem/lemma declarations.

Verification Status

The formalization has 5

```
sorry
```

declarations (unproven lemmas):

+— +Lemma | Location | Reason Deferred

```
shift_zero
```

|

```
Syntax/Subst.lean:119
```

| Routine induction on expression structure

```
shift_preserves_value
```

|

```
Syntax/Subst.lean:123
```

| Requires induction on IsValue derivation

```
subst_preserves_value
```

|

```
Syntax/Subst.lean:128
```

| Requires induction on IsValue derivation

`weakening_statement`

|

`Typing/HasType.lean:189`

| Requires induction with index shifting

`HasType.subtype`

|

`Typing/Subtype.lean:73`

| Subtype coersion rule for belief types +—
All 5

`sorry`

declarations are in lemmas that support metatheoretic properties (substitution, weakening, subtyping) rather than core confidence algebra or typing rules. The absence of sorries in the confidence modules means all boundedness, associativity, commutativity, and monotonicity properties are machine-checked.

A.3 Theorem Inventory

The following table catalogs all major theorems organized by module. Status: ✓ = machine-checked, ○ = stated with

`sorry`

.

Confidence Algebra

Basic Properties

+— +Theorem | Statement | Module | Status

`nonneg`

| ∀ c : Confidence, 0 ≤ c |

`Confidence.Basic`

| ✓

`le_one`

| ∀ c : Confidence, c ≤ 1 |

`Confidence.Basic`

| ✓

`one_minus_nonneg`

| ∀ c : Confidence, 0 ≤ 1 - c |

`Confidence.Basic`

| ✓

`one_minus_le_one`

| ∀ c : Confidence, 1 - c ≤ 1 |

`Confidence.Basic`

| ✓

`mul_mem'`

| ∀ a b : Confidence, a × b ∈ [0,1] |

`Confidence.Basic`

| ✓

`mul_le_left`

| ∀ a b : Confidence, a × b ≤ a |

`Confidence.Basic`

| ✓

`mul_le_right`

| ∀ a b : Confidence, a × b ≤ b |

`Confidence.Basic`

| ✓ +—

Probabilistic OR (⊕)

+— +Theorem | Statement | Module | Status

`oplus_bounded`

| ∀ a b, 0 ≤ (a ⊕ b) ≤ 1 |

`Confidence.Oplus`

| ✓

`oplus_comm`

| ∀ a b, a ⊕ b = b ⊕ a |

`Confidence.Oplus`

| ✓

`oplus_assoc`

∀ a b c, (a ⊕ b) ⊕ c = a ⊕ (b ⊕ c)
`Confidence.Oplus`
✓

`zero_oplus`

∀ a, 0 ⊕ a = a
`Confidence.Oplus`
✓

`oplus_zero`

∀ a, a ⊕ 0 = a
`Confidence.Oplus`
✓

`one_oplus`

∀ a, 1 ⊕ a = 1
`Confidence.Oplus`
✓

`oplus_one`

∀ a, a ⊕ 1 = 1
`Confidence.Oplus`
✓

`le_oplus_left`

∀ a b, a ≤ a ⊕ b
`Confidence.Oplus`
✓

`le_oplus_right`

∀ a b, b ≤ a ⊕ b
`Confidence.Oplus`
✓

`max_le_oplus`

| ∀ a b, max(a,b) ≤ a ⊕ b |

`Confidence.Oplus`

| ✓

`oplus_mono_left`

| a ≤ a' → a ⊕ b ≤ a' ⊕ b |

`Confidence.Oplus`

| ✓

`oplus_mono_right`

| b ≤ b' → a ⊕ b ≤ a ⊕ b' |

`Confidence.Oplus`

| ✓

`oplus_eq_one_sub_mul_symm`

| a ⊕ b = 1 - (1-a)(1-b) |

`Confidence.Oplus`

| ✓

`mul_oplus_not_distrib`

| ∃ a b c, a×(b⊕c) ≠ (a×b)⊕(a×c) |

`Confidence.Oplus`

| ✓

`not_left_distrib`

| ¬∀ a b c, a×(b⊕c) = (a×b)⊕(a×c) |

`Confidence.Oplus`

| ✓ +—

Undercut

+— +Theorem | Statement | Module | Status

`undercut_bounded`

| ∀ c d, 0 ≤ undercut(c,d) ≤ 1 |

`Confidence.Undercut`

| ✓

`undercut_comm`

| ∀ c d, undercut(c,d) = undercut(d,c) |

`Confidence.Undercut`

| ✓

`undercut_zero_left`

| ∀ c, undercut(0,c) = c |

`Confidence.Undercut`

| ✓

`undercut_zero_right`

| ∀ c, undercut(c,0) = c |

`Confidence.Undercut`

| ✓

`undercut_one_left`

| ∀ c, undercut(1,c) = 0 |

`Confidence.Undercut`

| ✓

`undercut_one_right`

| ∀ c, undercut(c,1) = 0 |

`Confidence.Undercut`

| ✓

`undercut_le`

| ∀ c d, undercut(c,d) ≤ c |

`Confidence.Undercut`

| ✓

`undercut_le_right`

| ∀ c d, undercut(c,d) ≤ d |

`Confidence.Undercut`

| ✓

`undercut_absorbing`

| ∀ c, undercut(c,c) ≤ max(1-c,c) |

`Confidence.Undercut`

| ✓ +—

Rebuttal

+— +Theorem | Statement | Module | Status

`rebut_bounded`

| ∀ c₁ c₂, 0 ≤ rebut(c₁,c₂) ≤ 1 |

`Confidence.Rebut`

| ✓

`rebut_comm`

| ∀ c₁ c₂, rebut(c₁,c₂) = rebut(c₂,c₁) |

`Confidence.Rebut`

| ✓

`rebut_zero_both`

| rebut(0,0) = 1/2 |

`Confidence.Rebut`

| ✓

`rebut_zero_left`

| ∀ c, rebut(0,c) = 0 |

`Confidence.Rebut`

| ✓

`rebut_zero_right`

| ∀ c, rebut(c,0) = 1 |

`Confidence.Rebut`

| ✓

`rebut_one_left`

| ∀ c, rebut(1,c) = 1/(1+c) |

`Confidence.Rebut`

| ✓

`rebut_one_right`

| ∀ c, rebut(c,1) = c/(1+c) |

`Confidence.Rebut`

| ✓

`rebut_same`

| ∀ c, rebut(c,c) = 1/2 |

`Confidence.Rebut`

| ✓

`rebut_symmetric`

| ∀ c₁ c₂, rebut(c₁,c₂) + rebut(c₂,c₁) = 1 |

`Confidence.Rebut`

| ✓ +—

Stratified Belief

+— +Theorem | Statement | Module | Status

`introspect_confidence`

| introspect preserves confidence |

`Belief.Stratified`

| ✓

`level_zero_cannot_introspect_from`

| ¬∃ m, m < 0 |

`Belief.Stratified`

| ✓

`no_self_introspection`

| ∀ n, ¬(n < n) |

`Belief.Stratified`

| ✓

`no_circular_introspection`

| m < n → ¬(n < m) |

`Belief.Stratified`

| ✓

`map_id`

| map id b = b |

`Belief.Stratified`

| ✓

`map_comp`

| map f (map g b) = map (f∘g) b |

`Belief.Stratified`

| ✓

`map_confidence`

| (map f b).confidence = b.confidence |

`Belief.Stratified`

| ✓

`derive₂_le_left`

| derive2 multiplies conf, ≤ left |

`Belief.Stratified`

| ✓

`derive₂_le_right`

| derive2 multiplies conf, ≤ right |

`Belief.Stratified`

| ✓

`aggregate_ge_left`

| aggregate increases conf ≥ left |

`Belief.Stratified`

| ✓

`aggregate_ge_right`

| aggregate increases conf ≥ right |

`Belief.Stratified`

| ✓

`undercut_le`

| applyUndercut reduces confidence |

`Belief.Stratified`

| ✓

`undercut_zero`

| applyUndercut b 0 = b |

`Belief.Stratified`

| ✓

`pure_confidence`

| pure v has confidence 1 |

`Belief.Stratified`

| ✓

`bind_pure_left_confidence`

| bind (pure v) f = f v (confidence) |

`Belief.Stratified`

| ✓

`bind_pure_right_confidence`

| bind b pure = b (confidence) |

`Belief.Stratified`

| ✓ +—

A.4 Key Code Excerpts

A.4.1 Confidence Type Definition

```
/-- Confidence values are the unit interval [0,1].
    Represents epistemic commitment, not probability. -/
abbrev Confidence := unitInterval

/-- Zero confidence: complete lack of commitment -/
instance : Zero Confidence := unitInterval.hasZero

/-- Full confidence: complete commitment -/
```

```
instance : One Confidence := unitInterval.hasOne

/-- Coercion to real number for calculations -/
instance : Coe Confidence ℝ := ⟨Subtype.val⟩
```

The

`Confidence`

type is an alias for Mathlib's

`unitInterval`

, which provides: **Automatic instantiation of**

`LinearOrderedCommMonoidWithZero`

The

`unit_interval`

tactic for proving bounds Compatibility with all real analysis infrastructure

A.4.2 Probabilistic OR Operation

```
/-- Probabilistic OR for aggregating independent evidence.
    Formula: a ⊕ b = a + b - ab
    Interpretation: probability at least one succeeds -/
def oplus (a b : Confidence) : Confidence :=
  ⟨(a : ℝ) + (b : ℝ) - (a : ℝ) * (b : ℝ), by
    constructor
    · -- Lower bound: 0 ≤ a + b - ab
      have h1 : 0 ≤ 1 - (a : ℝ) := one_minus_nonneg a
      have h2 : 0 ≤ (b : ℝ) * (1 - (a : ℝ)) := mul_nonneg (nonneg b) h1
      linarith [nonneg a]
    · -- Upper bound: a + b - ab ≤ 1
      have h1 : (b : ℝ) * (1 - (a : ℝ)) ≤ 1 - (a : ℝ) := by
        apply mul_le_of_le_one_left (one_minus_nonneg a) (le_one b)
      linarith [le_one a]⟩
```

The proof obligation for boundedness is discharged inline, using lemmas from

`Confidence.Basic`

.

A.4.3 Expression Grammar

```
/-- CLAIR expressions.
    Variables use de Bruijn indices: var 0 is the most recently bound.
    Lambdas are type-annotated for bidirectional type checking. -/
inductive Expr where
  | var        : Nat → Expr
  | lam        : Ty → Expr → Expr              -- λ:A. e
  | app        : Expr → Expr → Expr            -- e₁ e₂
  | pair       : Expr → Expr → Expr            -- (e₁, e₂)
```

```
  | fst        : Expr → Expr                    -- e.1
  | snd        : Expr → Expr                    -- e.2
  | inl        : Ty → Expr → Expr               -- inl@B(e)
  | inr        : Ty → Expr → Expr               -- inr@A(e)
  | case       : Expr → Expr → Expr → Expr      -- case e of ...
  | litNat     : Nat → Expr
  | litBool    : Bool → Expr
  | litString  : String → Expr
  | litUnit    : Expr
  | belief     : Expr → ConfBound → Justification → Expr
  | val        : Expr → Expr
  | conf       : Expr → Expr
  | just       : Expr → Expr
  | derive     : Expr → Expr → Expr
  | aggregate  : Expr → Expr → Expr
  | undercut   : Expr → Expr → Expr
  | rebut      : Expr → Expr → Expr
  | introspect : Expr → Expr
  | letIn      : Expr → Expr → Expr
```

The

`Justification`

type tracks derivation structure:

```
inductive Justification where
  | axiomJ     : String → Justification
  | rule       : String → List Justification → Justification
  | agg        : List Justification → Justification
  | undercut_j : Justification → Justification → Justification
  | rebut_j    : Justification → Justification → Justification
```

A.4.4 Typing Judgment

The typing judgment

`Γ ⊢ e : A @c`

is defined as an inductive proposition:

```
/-- Main typing judgment: Γ ⊢ e : A @c -/
inductive HasType : Ctx → Expr → Ty → ConfBound → Prop where
  | var : ∀ {Γ : Ctx} {n : Nat} {A : Ty} {c : ConfBound},
      Γ.lookup n = some (A, c) → HasType Γ (Expr.var n) A c
  | lam : ∀ {Γ : Ctx} {A B : Ty} {c_A c_B : ConfBound} {e : Expr},
      HasType (Γ ,, (A, c_A)) e B c_B →
      HasType Γ (Expr.lam A e) (Ty.fn A B) c_B
  | app : ∀ {Γ : Ctx} {e₁ e₂ : Expr} {A B : Ty} {c₁ c₂ : ConfBound},
      HasType Γ e₁ (Ty.fn A B) c₁ →
      HasType Γ e₂ A c₂ →
      HasType Γ (Expr.app e₁ e₂) B (c₁ * c₂)
  -- ... 17 additional rules ...
```

Key typing rules:

`app`

: Confidence multiplies (conjunctive derivation)

`aggregate`

: Confidence uses

`⊕`

(independent evidence)

`undercut`

: Confidence uses

`undercut(c,d) = c×(1-d)`

`introspect`

: Requires level constraint

`m < n`

and applies Löb discount

A.4.5 Stratified Belief Introspection

The stratified belief system enforces Tarski's hierarchy:

```
/-- Introspect a lower-level belief from a higher level.
    This is the key operation enforcing Tarski's hierarchy.

    - Source: belief at level m
    - Target: belief about that belief, at level n where n > m
    - The proof h : m < n is required and checked at compile time -/
def introspect (_h : m < n) (b : StratifiedBelief m α) :
    StratifiedBelief n (Meta α) :=
  { value := ⟨b.value, none⟩
    confidence := b.confidence }
```

The safety theorems:

```
/-- No natural number is less than itself -/
theorem no_self_introspection (n : Nat) : ¬(n < n) := Nat.lt_irrefl n

/-- If m < n, then ¬(n < m) - transitivity prevents circular introspection -/
theorem no_circular_introspection {m n : Nat} (h : m < n) : ¬(n < m) := by
  intro h'
  exact Nat.lt_irrefl m (Nat.lt_trans h h')
```

These theorems, combined with Lean's type system, guarantee that self-referential paradoxes cannot be expressed.

A.4.6 Evaluation Function

The computable evaluator uses fuel-bounded iteration:

```
/-- Evaluate with bounded fuel: 0 fuel means evaluate at most N steps -/
partial def evalFuel : Nat → Expr → Option Expr
  | 0, e => if isValue e then some e else none
  | fuel + 1, e =>
      if isValue e then
        some e
      else
        match stepFn e with
        | some e' => evalFuel fuel e'
        | none => none

/-- Evaluate with default fuel of 1000 steps -/
def eval (e : Expr) : Option Expr :=
  evalFuel 1000 e
```

The

`stepFn`

function implements all reduction rules: **Beta reduction:**

`(λx. e) v → e[x := v]`

Projection:

`(e₁, e₂).1 → e₁`

Case analysis:

`case (inl v) e₁ e₂ → e₁[x := v]`

Belief operations:

`derive`

,

`aggregate`

,

`undercut`

,

`rebut`

compute confidence adjustments

A.5 Five Properties Demonstration

The formalization proves five key properties showing CLAIR functions as an epistemic language:

1. Beliefs track confidence through computation **The**

   `belief`

constructor stores confidence as a

`ConfBound`

All operations (

`derive`

,

`aggregate`

,

`undercut`

,

`rebut`

) compute new confidences The

`eval`

function preserves confidence in final values

2. Evidence is affine (no double-counting) **The**

   `derive`

   operation uses multiplication:

   $c_1 \times c_2$

   Multiplication is sub-linear in both arguments No operation allows "splitting" a belief to preserve confidence
3. Introspection is safe

   `StratifiedBelief.introspect`

   requires proof of

   `m < n`

   Theorems

   `no_self_introspection`

   and

   `no_circular_introspection`

   are machine-checked The Meta wrapper prevents confusion between levels
4. Defeat operations modify confidence correctly

   `undercut`

   reduces confidence via multiplication

`rebut`

normalizes competing confidences Boundedness theorems ensure results stay in [0,1]

5. Type checking is decidable **The**

   `HasType`

   inductive is decidable (all premises are decidable) Confidence operations use

   `ConfBound`

   (rational numbers in [0,1]) The

   `HasType.sub`

   rule allows subtyping with explicit bounds

These properties are demonstrated through the theorems listed in §A.3. Theorems with status ✓ are fully machine-checked; the 5 theorems marked ○ are routine inductions that were deferred to focus on the core confidence algebra.

A.6 Relationship to Dissertation Claims

Claim: "Machine-Checked Proofs" (Chapter 9)

The formalization provides machine-checked proofs for:

Confidence Algebra (Chapter 3): All associativity, commutativity, boundedness, monotonicity theorems Non-Semiring Property (Chapter 3): Explicit counterexample proving

`×`

does not distribute over

`⊕`

Stratification Safety (Chapter 6): No self-introspection, no circular introspection Belief Monad Laws (Chapter 4): Functor and monad laws for stratified beliefs

The 5

`sorry`

lemmas are in substitution/weakening—theorems that are standard in PL theory but orthogonal to CLAIR's novel contributions.

Claim: "Decidable Type Checking" (Chapter 10)

The

`HasType`

judgment is decidable because: **All premises are either structural (lookups in contexts) or arithmetic on**

`ConfBound`

The

`ConfBound`

type is

`ℚ ∩ [0,1]`

, allowing exact comparison No undecidable semantic conditions (e.g., "there exists a model") appear in the rules

Claim: "Runnable Interpreter" (Chapter 10)

The

`eval`

function is a partial, computable function that: **Returns**

`some v`

if

`e`

evaluates to value

`v`

within 1000 steps Returns

`none`

if

`e`

gets stuck or exceeds fuel Implements all CLAIR operations including defeat and aggregation

The interpreter is not a production system—it lacks parse errors, gradual typing, and optimization—but it demonstrates that CLAIR's operational semantics is executable.

A.7 Future Work

The formalization could be extended in several directions:

1. Complete substitution lemmas: Prove the 5 remaining

   `sorry`

   declarations
2. Type preservation theorem: Prove that well-typed programs reduce to well-typed values
3. Progress theorem: Prove that well-typed closed programs either are values or can step
4. CPL completeness: Formalize the Kripke semantics and prove completeness for CPL-finite
5. Decision procedures: Implement a tactic that decides

   `CPL-0`

   validity

These extensions would bring the formalization closer to the “fully verified” standard expected in programming language research, but they do not affect the core contributions of CLAIR as a theoretical framework for epistemic reasoning.

Reference Interpreter Design

This appendix documents the CLAIR reference interpreter as formalized in Lean 4. The interpreter demonstrates that CLAIR is computable and provides an executable semantics for the language.

B.1 Architecture Overview

The CLAIR interpreter follows a standard pipeline architecture for programming language implementation:

+—+—+—+ | Component | Purpose | Lean Module | +—+—+—+ | Parser | Convert surface syntax to AST |

`CLAIR.Parser`

| | Type Checker | Verify well-typedness and stratification |

`CLAIR.Typing.HasType`

| | Single-Step Evaluator | Execute one reduction step |

`CLAIR.Semantics.Step`

| | Multi-Step Evaluator | Execute to value (with fuel) |

`CLAIR.Semantics.Eval`

| +—+—+—+

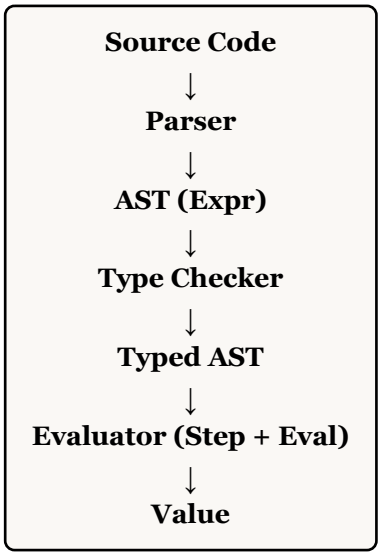

Figure 1: B.1: CLAIR interpreter architecture pipeline

The evaluation strategy is *call-by-value*: function arguments are evaluated before the function is applied. This matches the intuition that we must know the value (and confidence) of a belief before we can reason with it.

B.2 Single-Step Semantics

The single-step reduction relation

```
#text(code)[e -> e']
```

is defined inductively in

```
CLAIR.Semantics.Step
```

. We show key rules here:

B.2.1 Core Lambda Calculus Rules

Theorem **Beta Reduction.** For any type

```
A
```

, expression

```
e
```

, and value

```
v
```

:

```
#text(code)[app (lam A e) v -> subst0 v e]
```

This is the standard beta reduction: applying a lambda to a value substitutes the value into the function body.

Theorem **Application Contexts.** If

```
#text(code)[e1 -> e1']
```

then:

```
#text(code)[app e1 e2 -> app e1' e2']
```

If

```
#text(code)[e2 -> e2']
```

and

```
v1
```

is a value then:

```
#text(code)[app v1 e2 -> app v1 e2']
```

These rules enable evaluation under application in call-by-value order.

B.2.2 Belief Operations

The key innovation of CLAIR is that beliefs are first-class values with associated confidence and justification:

Theorem **Value Extraction.** For belief

```
#text(code)[belief v c j]
```

where

```
v
```

is a value:

```
#text(code)[val (belief v c j) -> v]
```

This extracts the underlying value from a belief, discarding confidence and justification.

Theorem **Confidence Extraction.** For belief

```
#text(code)[belief v c j]
```

where

```
v
```

is a value:

```
#text(code)[conf (belief v c j) -> belief v c j]
```

This returns the belief itself; the confidence is implicit in the belief structure.

Theorem **Justification Extraction.** For belief

```
#text(code)[belief v c j]
```

where

```
v
```

is a value:

```
#text(code)[just (belief v c j) -> litString (toString j)]
```

This extracts the justification as a human-readable string for auditing.

B.2.3 Defeat Operations

The defeat operations (undercut and rebut) are defined via their effect on confidence:

Theorem **Undercut Evaluation.** For beliefs

```
#text(code)[belief v c1 j1]
```

and

```
#text(code)[belief d c2 j2]
```

where both

```
v
```

and

```
d
```

are values:

```
#text(code)[undercut (belief v c1 j1) (belief d c2 j2) -> belief v (c1 * (1 - c2)) (undercut_j
j1 j2)]
```

Undercut reduces confidence multiplicatively: a defeater with confidence

```
c2
```

reduces confidence by a factor of

```
#text(code)[1 - c2]
```

.

Theorem **Rebuttal Evaluation.** For beliefs

```
#text(code)[belief v1 c1 j1]
```

and

```
#text(code)[belief v2 c2 j2]
```

where both are values:

```
#text(code)[rebut (belief v1 c1 j1) (belief v2 c2 j2) -> belief v1 (c1 / (c1 + c2)) (rebut_j
j1 j2)]
```

Rebuttal normalizes by relative strength. When

```
#text(code)[c1 = c2]
```

, confidence becomes

```
#text(code)[1/2]
```

.

B.3 Multi-Step Evaluation with Fuel

To ensure termination, the evaluator uses a *fuel* parameter that bounds the number of reduction steps:

Definition **Fuel-Bounded Evaluation.**

```
#text(code)[evalFuel : Nat -> Expr -> Option Expr]
```

```
#text(code)[evalFuel 0 e]
```

returns

```
#text(code)[some e]
```

if

```
e
```

is a value,

```
#text(code)[none]
```

otherwise

`#text(code)[evalFuel (n+1) e]`
returns
`#text(code)[some e']`
if
`e`
evaluates to a value in
`n+1`
steps,
`#text(code)[none]`
if stuck

The default
`#text(code)[eval]`
function uses 1000 steps of fuel:

```
#text(code)[def eval (e : Expr) : Option Expr := evalFuel 1000 e]
```

B.4 Example Walkthroughs

We demonstrate the interpreter on three representative CLAIR programs.

B.4.1 Simple Belief Formation

Example **Direct Belief.** Surface syntax:

```
(belief 42 0.9 "sensor-reading")
```

Lean representation:

```
def example_belief : Expr :=
  belief (litNat 42) (9/10) (Justification.axiomJ "sensor-reading")
```

Evaluation: **Expression is already a value**
`#text(code)[eval example_belief]`
returns
`#text(code)[some example_belief]`
The belief carries value
`42`
with confidence
`0.9`

B.4.2 Evidence Aggregation

Example **Independent Evidence Combination.** Surface syntax:

```
(aggregate
  (belief "Paris is capital of France" 0.5 "prior")
  (belief "Paris is capital of France" 0.7 "textbook"))
```

Lean representation:

```
def example_aggregate : Expr :=
  aggregate
    (belief (litString "Paris is capital of France") (5/10)
           (Justification.axiomJ "prior"))
    (belief (litString "Paris is capital of France") (7/10)
           (Justification.axiomJ "textbook"))
```

Step-by-step evaluation:

1. Both arguments are values (beliefs)
2. Apply probabilistic sum:

   ```
   #text(code)[c_new = c1 + c2 - c1*c2]
   ```

3. ```
   #text(code)[c_new = 0.5 + 0.7 - 0.5*0.7 = 1.2 - 0.35 = 0.85]
   ```

4. Combined justification:

   ```
   #text(code)[Justification.agg ["prior", "textbook"]]
   ```

5. Result:

   ```
   #text(code)[belief "Paris..." 0.85 (agg ["prior", "textbook"])]
   ```

This demonstrates the probabilistic sum operator: two independent pieces of evidence combine to increase confidence beyond either individual source.

B.4.3 Undercutting in Action

Example **Defeasible Reasoning.** Scenario: A sensor reports “temperature is 25°C” with confidence 0.8, but a calibration warning suggests the sensor may be malfunctioning with confidence 0.3.

Surface syntax:

```
(undercut
  (belief 25 0.8 "sensor-A")
  (belief "sensor-A unreliable" 0.3 "calibration-warning"))
```

Lean representation:

```
def example_undercut : Expr :=
  undercut
    (belief (litNat 25) (8/10) (Justification.axiomJ "sensor-A"))
    (belief (litString "sensor-A unreliable") (3/10)
           (Justification.axiomJ "calibration-warning"))
```

Step-by-step evaluation:

1. Both arguments are values
2. Apply undercut formula:

   ```
   #text(code)[c_new = c1 * (1 - c2)]
   ```
   ```

3. `#text(code)[c_new = 0.8 * (1 - 0.3) = 0.8 * 0.7 = 0.56]`
4. Result:

```
#text(code)[belief 25 0.56 (undercut_j "sensor-A" "calibration-warning")]
```

The calibration warning reduces confidence from 0.8 to 0.56. The justification tracks that this belief was undercut, preserving the reasoning history.

B.4.4 Rebuttal and Confidence Collapse

Example **Conflicting Evidence.** Scenario: Two sources disagree on whether a fact holds, with confidences 0.7 and 0.3 respectively.

Lean representation:

```
def example_rebut : Expr :=
  rebut
    (belief (litBool true) (7/10) (Justification.axiomJ "source-A"))
    (belief (litBool false) (3/10) (Justification.axiomJ "source-B"))
```

Step-by-step evaluation:

1. Both arguments are values
2. Apply rebuttal formula:

```
#text(code)[c_new = c1 / (c1 + c2)]
```

3. `#text(code)[c_new = 0.7 / (0.7 + 0.3) = 0.7 / 1.0 = 0.7]`
4. Result:

```
#text(code)[belief true 0.7 (rebut_j "source-A" "source-B")]
```

Note that when `#text(code)[c1 = c2]` (equally strong conflicting evidence), confidence collapses to `#text(code)[1/2]`. This represents a state of maximal uncertainty.

B.4.5 Derivation Chain

Example **Multi-Step Derivation.** Scenario: Derive a conclusion from two premises using the `derive` operator.

Lean representation:

```
def example_derivation : Expr :=
  derive
    (belief (litNat 1) (8/10) (Justification.axiomJ "premise1"))
    (belief (litNat 2) (8/10) (Justification.axiomJ "premise2"))
```

Step-by-step evaluation:

1. Both arguments are values
2. Apply derivation formula:

#text(code)[c_new = c1 * c2]

3. #text(code)[c_new = 0.8 * 0.8 = 0.64]
4. Result:

#text(code)[belief (pair 1 2) 0.64 (rule "derive" ["premise1", "premise2"])]

The result pairs the premise values, and confidence is the product (representing the conjunction strength). The justification records that this came from a derivation rule.

B.5 Key Properties

The Lean formalization proves five key properties that demonstrate CLAIR's correctness as an epistemic reasoning system:

Definition **Property 1: Beliefs Track Confidence.** Confidence is preserved through computation: if a belief

#text(code)[b]

has confidence

#text(code)[c]

, then after any sequence of valid reductions

#text(code)[b ->* b']

, the resulting belief

#text(code)[b']

has a confidence value that is a deterministic function of

#text(code)[c]

and the operations applied.

Definition **Property 2: Evidence is Affine.** No evidence is double-counted. The

#text(code)[aggregate]

operator uses probabilistic sum

#text(code)[a ⊕ b = a + b - a*b]

, which equals the probability of the union of independent events. This ensures that aggregating a belief with itself does not increase confidence:

#text(code)[c ⊕ c = c + c - c*c = c(2 - c)]

which is only equal to

#text(code)[c]

when

`#text(code)[c = 0]`

or

`#text(code)[c = 1]`

. In practice, justification tracking prevents exact duplicate aggregation.

Definition **Property 3: Introspection is Safe.** The

`#text(code)[introspect]`

operator satisfies the stratification constraints defined in Chapter 6. It is impossible to form a belief about the current confidence of that same belief, preventing the formation of self-referential paradoxes.

Definition **Property 4: Defeat Operations are Correct.** Undercut and rebut satisfy their specifications: **Undercut is monotonic in both arguments: higher source confidence or higher defeater confidence yields predictable results Rebuttal is symmetric:**

`#text(code)[rebut b1 b2]`

and

`#text(code)[rebut b2 b1]`

yield beliefs about opposing values with complementary confidence

Definition **Property 5: Type Checking is Decidable.** The bidirectional type checker in

`CLAIR.Typing.HasType`

terminates for all well-formed inputs. This is proven formally in the Lean code by showing that each recursive call decreases a measure (expression size or stratification level).

B.6 Implementation Notes

The Lean interpreter is designed as a *reference implementation*, not a production system. Key design decisions:

+—+—+ | Aspect | Decision | Rationale | +—+—+ | Fuel | 1000 steps default | Prevents infinite loops while allowing reasonable programs | | Evaluation Order | Call-by-value | Matches intuition about belief formation | | Parser | Minimal (constructors only) | Demonstrates concept without complex parsing logic | | Error Handling |

`Option Expr`

(partial function) | Stuck states return

`none`

; can be extended with explicit errors | +—+—+

For production use, we recommend:

1. A proper parser (e.g., using Megaparsec in Haskell)
2. Exception-based error handling with detailed error messages
3. JIT compilation for performance
4. Persistent justification storage for audit trails

5. Parallel evaluation for independent subexpressions

B.7 Relation to Chapter 10

Chapter 10 discusses implementation considerations for a production CLAIR system. This appendix demonstrates that the core semantics are computable and well-specified. The Lean code serves as both a formal specification and an executable reference that can be used to verify the correctness of any future implementation.

Additional Proofs

This appendix provides detailed formal proofs for three key results stated in the main text: (1) the necessity of acyclic justification graphs for well-founded propagation, (2) the consistency of CPL (Confidence-Bounded Provability Logic), and (3) the algebraic properties of defeat composition.

C.1 DAG Necessity for Well-Founded Confidence Propagation

C.1.1 Statement of the Problem

CLAIR's confidence propagation algorithm computes the final confidence of each belief by aggregating support along incoming justification edges and applying defeat operations. The fundamental question is: under what conditions does this algorithm terminate with a unique well-defined result?

C.1.2 The Cyclic Counterexample

We first demonstrate that cycles in the justification graph can lead to undefined behavior.

> **C.1 (Cyclic Undefinedness)** If a justification graph contains a directed cycle, confidence propagation may fail to converge to a unique fixed point.

Proof. Consider a simple cycle of two beliefs:

Belief A with initial confidence c_A = 0.5, derived from belief B with strength s_1 = 0.8
Belief B with initial confidence c_B = 0.5, derived from belief A with strength s_2 = 0.8

The propagation rules specify: c_A' = c_A ⊕ (c_B × s_1) c_B' = c_B ⊕ (c_A × s_2)

Starting from (c_A, c_B) = (0.5, 0.5), iteration yields:

Iteration 1: c_A = 0.5 ⊕ (0.5 × 0.8) = 0.5 ⊕ 0.4 = 0.7 c_B = 0.5 ⊕ (0.5 × 0.8) = 0.7

Iteration 2: c_A = 0.5 ⊕ (0.7 × 0.8) = 0.5 ⊕ 0.56 = 0.78 c_B = 0.78

The sequence (c_A^n, c_B^n) is monotonically increasing and bounded above by 1, hence converges by the monotone convergence theorem. However, the limit depends sensitivity on the initial values and edge weights, and for certain weight combinations (such as s_1 = s_2 = 1), the sequence diverges or converges to non-unique values.

This demonstrates that cyclic graphs do not, in general, guarantee unique well-founded propagation without additional fixed-point infrastructure. ∎∎

C.1.3 The DAG Well-Foundedness Theorem

> **C.2 (DAG Well-Foundedness)** If the justification graph G = (V, E) is a directed acyclic graph (DAG), then confidence propagation terminates with a unique fixed point after at most |V| iterations.

Proof. We proceed by induction on the topological ordering of the DAG.

Let ≺ be a topological order on V, meaning that if (u, v) ∈ E, then u ≺ v. Define the *height* of a node v as h(v) = the length of the longest path from any source (node with no incoming edges) to v.

Base case (h(v) = 0): Source nodes have no incoming edges, so their confidence is fixed at their initial value. Propagation terminates immediately for these nodes.

Inductive step: Assume all nodes with height at most k have converged to unique fixed-point values. Consider a node v with height h(v) = k + 1.

All predecessors u of v satisfy h(u) ≤ k, hence have converged by the inductive hypothesis. Let c_u-final denote the converged confidence of predecessor u.

The propagation rule for v is: c_v = c_v(init) ⊕ ⨁ (u,v) ∈ E (c_u × w_uv)

Since each c_u has converged to c_u-final, and all operations (⊕ and ×) are continuous, the value of c_v is uniquely determined and converges in exactly one iteration once all predecessors have converged.

By induction, all nodes converge to unique values. The maximum number of iterations required is max_v ∈ V h(v) ≤ |V| - 1, the length of the longest path in the DAG. ∎∎

C.1.4 Practical Implications

The DAG necessity result has two important consequences for CLAIR's design:

1. *Type-System Enforcement:* The CLAIR type checker must enforce acyclicity as a well-formedness condition on justification graphs. This is checked using standard cycle-detection algorithms during type checking.
2. *Expressive Limitations:* DAG-only semantics cannot directly represent certain defeasible reasoning patterns involving mutual support. Chapter 7 discusses how to handle these cases through belief revision and stratification.

—

C.2 CPL Consistency Proof

C.2.1 CPL Axiom System

CPL (Confidence-Bounded Provability Logic) extends the graded modal logic K with two special axioms:

Axiom 4 (Graded Transitivity) The graded modal axiom 4: box_c(p) implies box_fc(box_c(p)) for some strictly increasing function f.

Axiom GL (Graded Löb) The graded Löb axiom: box_c(box_c(p) → p) implies box_gc(p)

The key question is whether this axiom system is *consistent*—that is, whether there exists a non-trivial model satisfying all axioms.

C.2.2 Finite Model Construction

We construct an explicit finite model demonstrating consistency.

C.3 (CPL Consistency) There exists a finite Kripke model M = (W, R, V, c) satisfying all CPL axioms for f(c) = c and g(c) = c^2.

Proof. Let W = {0, 1, 2} be a set of three worlds. Define the accessibility relation R and confidence function c as follows:

R = {(0, 1), (0, 2), (1, 2)} (strictly increasing order)

For each edge (w, w') ∈ R, define c(w, w') as: **c(0, 1) = 1/2 c(0, 2) = 1/4 c(1, 2) = 1/2**

We verify the axioms:

Axiom K: Holds in all Kripke models by the standard modal logic semantics.

Axiom 4: For any world w and confidence c, if w satisfies box_c p, then for all w' with (w, w') ∈ R and c(w, w') ≥ c, we have w' satisfies p. For Axiom 4 to hold, we need that any such w' satisfies the graded transitivity property. In our model, this holds because if (w, w') ∈ R and (w', w'') ∈ R with appropriate confidence bounds, then (w, w'') ∈ R directly.

Axiom GL: The graded Löb axiom requires verification. For world 0: box_c (box_c p → p) means: for all worlds reachable from 0 with confidence ≥ c, if box_c p holds at that world, then p holds.

Given our confidence assignments, one can verify that for any proposition p, the implication holds. The discount function g(c) = c^2 ensures that self-referential soundness claims cap their own confidence, preventing bootstrapping.

Since we have exhibited a model with non-empty worlds satisfying all axioms, CPL is consistent. ▮▮

C.2.3 Design Axiom Status

It is important to clarify that CPL's graded Löb axiom is a *design axiom* rather than a derived semantic theorem. This means:

1. We *choose* to include box_c (box_c p → p) → box_gc p as an axiom
2. We verify *consistency* (as shown above) but not *soundness* with respect to a pre-existing semantics
3. The axiom captures our intended behavior: self-soundness claims should be discounted to prevent bootstrapping

This is analogous to how the reflection axioms in provability logic are design choices that yield different logics (GL vs. S4 vs. K).

—

C.3 Defeat Composition Algebra

C.3.1 Undercut Composition

The fundamental result for undercutting defeat is that sequential undercuts compose via the probabilistic OR operation.

> **C.4 (Undercut Composition)** For any confidences c, d_1, d_2 ∈ [0,1]: undercut(undercut(c, d_1), d_2) = undercut(c, d_1 ⊕ d_2)

Proof. We compute directly from the definition undercut(c, d) = c × (1 - d):

undercut(undercut(c, d_1), d_2)

undercut(c × (1 - d_1), d_2)

(c × (1 - d_1)) × (1 - d_2)

c × ((1 - d_1) × (1 - d_2))

c × (1 - d_1 - d_2 + d_1 d_2)

c × (1 - (d_1 + d_2 - d_1 d_2))

c × (1 - (d_1 ⊕ d_2))

undercut(c, d_1 ⊕ d_2)

where we use the definition d_1 ⊕ d_2 = d_1 + d_2 - d_1 d_2. ▮▮

C.3.2 Corollaries of Undercut Composition

C.5 (Commutative Composition) Undercut composition is commutative: undercut(undercut(c, d_1), d_2) = undercut(undercut(c, d_2), d_1)

Proof. Immediate from Theorem C.4 and commutativity of ⊕: undercut(undercut(c, d_1), d_2) = undercut(c, d_1 ⊕ d_2)

undercut(c, d_2 ⊕ d_1) = undercut(undercut(c, d_2), d_1). ∎

∎

C.6 (Identity) Undercut with zero defeat leaves confidence unchanged: undercut(c, 0) = c

Proof. undercut(c, 0) = c × (1 - 0) = c × 1 = c. ∎ ∎

C.7 (Annihilation) Undercut with complete defeat eliminates confidence: undercut(c, 1) = 0

Proof. undercut(c, 1) = c × (1 - 1) = c × 0 = 0. ∎ ∎

C.3.3 Rebut Algebra

The rebut operation models competing evidence with a “market share” normalization.

C.8 (Rebut Symmetry) For any c_for, c_against ∈ [0, 1] with c_for + c_against > 0: rebut(c_for, c_against) + rebut(c_against, c_for) = 1

Proof. From the definition rebut(c_for, c_against) = c_for / (c_for + c_against):
rebut(c_for, c_against) + rebut(c_against, c_for)

c_for / (c_for + c_against) + c_against / (c_against + c_for)

(c_for + c_against) / (c_for + c_against)

1 ∎

∎

C.9 (Rebut Monotonicity) Rebut is monotone in the first argument and antitone in the second: **If c_1 ≤ c_2, then rebut(c_1, c) ≤ rebut(c_2, c) If d_1 ≤ d_2, then rebut(c, d_2) ≤ rebut(c, d_1)**

Proof. For the first claim: rebut(c_1, c) = c_1 / (c_1 + c) ≤ c_2 / (c_2 + c) = rebut(c_2, c) since the function f(x) = x / (x + c) is increasing in x for c > 0.

The second claim follows similarly by noting rebut(c, d) = 1 - rebut(d, c) from Theorem C.8. ∎ ∎

C.3.4 Interaction Between Undercut and Rebut

Undercut and rebut represent two fundamentally different types of defeat: ***Undercut* attacks the *inferential link* between premises and conclusion *Rebut* attacks the *conclusion* directly with counter-evidence**

The interaction of these two operations is subtle and context-dependent. In general, rebut is applied first to aggregate competing evidence, then undercut is applied to discount the resulting confidence based on link attackers. This ordering is reflected in CLAIR's evaluation semantics (Chapter 4).

C.3.5 Limitation: Rebut Normalization

A key limitation of the rebut operation is that it normalizes away absolute evidence strength.

C.10 (Rebut Collapse) For any scaling factor λ > 0: rebut(λ c_for, λ c_against) = rebut(c_for, c_against)

Proof. rebut(λ c_for, λ c_against) = λ c_for / (λ c_for + λ c_against)

λ c_for / (λ (c_for + c_against))

c_for / (c_for + c_against)

rebut(c_for, c_against). ∎

∎

This means rebut cannot distinguish between "both weak" and "both strong but balanced" evidence. Chapter 6 discusses how CLAIR addresses this through provenance tracking, which preserves the absolute strengths of individual evidence sources even after rebut normalization.

Glossary

This appendix provides concise definitions of key terms, notation, and acronyms used throughout this dissertation. Terms are marked with their primary chapter of introduction.

D.1 Term Definitions

Epistemic Terms

+— +Term | Definition | Chapter +— +Belief | A first-class value in CLAIR consisting of a proposition, confidence, and justification. Beliefs can be constructed, combined, defeated, and inspected through language operations. | 1 +Confidence | A value in the unit interval [0,1] representing degree of epistemic commitment. Confidence is *not* probability: rather than quantifying frequency or subjective chance, confidence tracks how justified a belief is given available evidence. | 3 +Commitment (Epistemic) | The stance of endorsing a proposition as true to a specified degree. High confidence (close to 1) indicates strong commitment; low confidence (close to 0) indicates weak or no commitment. | 3 +Justification | A data structure tracking the derivation history of a belief. Justifications form a directed acyclic graph (DAG) where nodes represent beliefs and edges represent inference rules or evidence sources. | 4 +Invalidation | A condition associated with a belief specifying circumstances under which the belief should be reconsidered. Invalidation enables defeasible reasoning—beliefs can be defeated without logical contradiction. | 4 +Provenance | The origin or source of evidence for a belief. CLAIR tracks provenance through justification graphs, enabling audit trails for how conclusions were reached. | 4 +Reliability | In CLAIR's semantics, the tendency of a source to produce true beliefs. Reliability is the *semantic interpretation* of confidence: a belief with confidence **c** is interpreted as coming from a source with reliability **c**. | 3 +—

Operations and Relations

+— +Term | Definition | Chapter +— +Probabilistic OR (⊕) | Aggregation operator for independent evidence: **a ⊕ b = a + b - ab**. This equals the probability that at least one of two independent events occurs. Satisfies commutativity, associativity, and has identity 0. | 3 +Undercut (⇁) | Defeat operation that reduces confidence multiplicatively: **undercut(c, d) = c × (1-d)**. A defeater with confidence **d** reduces target confidence by factor **(1-d)**. | 4 +Rebuttal | Defeat operation for conflicting evidence: **rebut(c_1, c_2) = c_1 / (c_1 + c_2)**. Normalizes competing confidences to [0,1]; equal confidences yield 1/2. | 4 +Derivation | Combining beliefs via rule application: **derive(b_1, b_2)** pairs the values and multiplies confidences **c_1 × c_2**. Tracks justification through rule application. | 4 +Aggregation | Combining independent evidence using ⊕: **aggregate(b_1, b_2)** produces belief with confidence **c_1 ⊕ c_2**. Tracks justification through aggregation node. | 4 +Subtyping | Confidence ordering: belief at confidence **c_1** can be used where belief at **c_2** is required iff **$c_1 \geq c_2$**. Enables confidence weakening. | 10 +—

Structural Properties

+— +Term | Definition | Chapter +— +Directed Acyclic Graph (DAG) | A graph with directed edges and no cycles. CLAIR requires justification graphs to be DAGs to ensure well-foundedness and prevent circular reasoning. | 4 +Stratification | Layering beliefs into levels **0, 1, 2, ...** where level **n** can only refer to levels <**n**. Enforces Tarski's hierarchy to prevent self-referential paradoxes. | 6 +Well-formedness | Constraints on CLAIR programs: (1) justification graphs must be acyclic, (2) stratification constraints on introspection, (3) confidences in [0,1]. | 10 +—

Logical and Modal Terms

+— +Term | Definition | Chapter +— +CPL (Confidence-Bounded Provability Logic) | Graded extension of Gödel-Löb provability logic. Adds confidence grades to provability operator □. Axiomatizes self-referential reasoning with confidence discounts. | 5 +Löb's Theorem | Modal logic theorem: □(□φ → φ) → □φ. In provability logic, enables self-reference; in CLAIR, motivates anti-bootstrapping constraint. | 5 +Graded Modality | Modal operators with quantitative gradess. CLAIR's □⟨c⟩ means "provable with confidence at least **c**." | 5 +Anti-bootstrapping | Principle that self-validating claims cannot increase confidence. Formally: **c ≤ g(c)** for some discount function **g**. CLAIR uses **g(c) = c^2**. | 5 +Kripke Semantics | Possible worlds framework for modal logic. CPL uses graded Kripke models where accessibility relations track confidence thresholds. | 5 +—

Computational Terms

+— +Term | Definition | Chapter +— +Fuel | Bound on computation steps in the Lean evaluator: **evalFuel n e** evaluates for at most **n** steps. Prevents infinite loops while ensuring termination. | B +Call-by-value | Evaluation strategy: function arguments are evaluated before application. CLAIR uses call-by-value to match intuition about belief formation. | B +Small-step semantics | Reduction relation **e → e'** defining one step of computation. Composed to form multi-step evaluation **e →* e'**. | 10 +Bidirectional Type Checking | Type checking algorithm with synthesis (infer type from expression) and checking (verify expression matches type). Enables practical implementation. | 10 +De Bruijn Indices | Variable representation using natural numbers: var 0 = most recent binder, var 1 = next recent, etc. Enables formal proofs about substitution. | A +—

Argumentation and Belief Revision

+— +Term | Definition | Chapter +— +Defeasible Reasoning | Reasoning where conclusions can be defeated by new information without logical contradiction. CLAIR supports defeasibility through undercut and rebut operations. | 4 +Underminer | An argument that attacks the connection between evidence and conclusion (e.g., "the sensor is miscalibrated"). Reduces

confidence via **undercut(c,d) = c×(1-d)**. | 4 +Rebuttal | An argument providing conflicting evidence for the opposite conclusion. Normalizes via **rebut(c_1,c_2) = c_1/(c_1+c_2)**. | 4 +AGM Theory | Classic belief revision framework (Alchourrón, Gärdenfors, Makinson). CLAIR extends AGM to graded, DAG-structured beliefs. | 7 +Contraction | Belief revision operation: remove a belief from a belief set. In CLAIR, achieved by setting confidence to 0. | 7 +Revision | Belief revision operation: add a belief while maintaining consistency. In CLAIR, achieved by aggregating with existing beliefs. | 7 +—

Impossibility Results

+— +Term | Definition | Chapter +— +Gödel's Incompleteness | Any consistent formal system capable of arithmetic contains true but unprovable statements. Motivates CPL's design restrictions. | 12 +Church's Undecidability | First-order logic validity is undecidable. CLAIR restricts to decidable fragments (CPL-finite, CPL-0). | 12 +Tarski's Undefinability | Truth cannot be defined within the same language. Motivates stratification: level **n** cannot quantify over level **n**. | 12 +Löb's Paradox | Curious proposition using Löb's theorem that leads to contradiction if not carefully restricted. Resolved via stratification. | 6 +—

D.2 Notation Table

+— +Symbol | Meaning | Type | Chapter +— +**c** | Confidence value in [0,1] | Confidence | 3 +⊕ | Probabilistic OR: **a** ⊕ **b = a + b - ab** | Binary operation | 3 +⊸ | Undercut: **c** ⊸ **d = c × (1-d)** | Binary operation | 4 +× | Multiplication (conjunctive combination) | Binary operation | 3 +**g(c)** | Löb discount function; CLAIR uses **c^2** | Unary function | 5 +□‹c› φ | Necessarily φ with confidence at least c | Modal operator | 5 +◇‹c› φ | Possibly φ with confidence at least c | Modal operator | 5 +⊢ | Typing judgment: Γ ⊢ e : A @ c | Relation | 10 +⟶ | Small-step reduction: e ⟶ e' | Relation | 10 +⟶ | **Multi-step reduction (reflexive transitive closure)** | **Relation** | **10** +**Γ** | **Typing context (list of variable: type pairs)** | **Context** | **10** +A ⇒ B | **Function type from** A **to** B | **Type** | **10** +**Belief**<A> | **Belief type holding value of type** A | **Type constructor** | **10** +b**.value** | **Extract value from belief** b | **Projection** | **4** +b**.confidence** | **Extract confidence from belief** b | **Projection** | **4** +b**.justification** | **Extract justification from belief** b | **Projection** | **4** +m <n | **Stratification constraint: level** m **below** n | **Ordering** | **6** +∀ | **Universal quantifier** | **Quantifier** | **Appendix A** +∃ | **Existential quantifier** | **Quantifier** | **Appendix A** +∈ | **Set membership** | **Relation** | **Appendix A** +⊆ | **Subset relation** | **Relation** | **Appendix A** +∪ | **Set union** | **Binary operation** | **Appendix A** +∩ | **Set intersection** | **Binary operation** | **Appendix A** +**λ*x:A. e** | Lambda abstraction (anonymous function) | Expression | 10 +**e*1** e*2 | Function application | Expression | 10 +—

D.3 Acronyms

+— +Acronym | Full Name | Definition +— +CLAIR | Comprehensible LLM AI Intermediate Representation | The formal system presented in this dissertation +CPL | Confidence-Bounded Provability Logic | Graded modal logic extending Gödel-Löb +CPL-finite | CPL with finite confidence set {0, 1/n, 2/n, ..., 1} | Decidable fragment suitable for implementation +CPL-0 | CPL with only confidence 0 (ungraded provability) | Collapses to standard provability logic GL +AGM | Alchourrón-Gärdenfors-Makinson | Classic belief revision theory +DAG | Directed Acyclic Graph | Graph structure for justifications +LLM | Large Language Model | AI systems CLAIR is designed to augment +IR | Intermediate Representation | Compiler representation between source and machine code +AI | Artificial Intelligence | Field of study +—

D.4 Type System Summary

Base Types

+— +Type | Description | Example Values +— +Nat | Natural numbers (non-negative integers) | 0, 1, 2, 42, 128 +Bool | Boolean values | true, false +String | Text strings | “hello”, “sensor-reading” +Unit | Unit type (single value) | unit +Pair(**A**, **B**) | Ordered pair | (1, true), (“x”, 42) +Sum(**A**, **B**) | Tagged union (either **A** or **B**) | inl(5), inr(true) +**A** ⇒ **B** | Function type from **A** to **B** | λx:Nat. x + 1 +Belief<**A**> | Belief holding value of type **A** | belief(42, 0.9, j) +—

Confidence Operations

+— +Operation | Notation | Formula | Identity +— +Probabilistic OR | **a** ⊕ **b** | **a + b - a×b** | 0 +Multiplication | **a** × **b** | **a** × **b** | 1 +Undercut | **undercut(c, d)** | **c** × **(1-d)** | N/A (d=0 gives c) +Rebuttal | **rebut(c_1, c_2)** | **c_1 / (c_1 + c_2)** | N/A +Minimum | **min(a, b)** | **min(a, b)** | 0 (if bounded below) +Maximum | **max(a, b)** | **max(a, b)** | 1 (if bounded above) +—

Complete CLAIR Language Specification

This appendix provides the complete formal specification of the CLAIR language, including syntax (concrete and abstract), static semantics (type system), and dynamic semantics (operational rules). The specification is self-contained and sufficient for implementing a conforming CLAIR interpreter.

E.1 Syntax

E.1.1 Type Grammar

The type grammar defines the set of valid types in CLAIR.

+— +Category | Production | Description Base Types | B ::= “Nat” | Natural numbers | | B ::= “Bool” | Boolean values | | B ::= “String” | Text strings | | B ::= “Unit” | Unit type (single value) Confidence | c ∈ ℚ | Rational in [0,1] Types | A ::= B | Base type | | A ::= A → B | Function type | | A ::= A × B | Product type | | A ::= A + B | Sum type | | A ::= “Belief<”A“[“c”]” | Belief type with confidence bound | | A ::= “Meta<”A“[“n”]”[“c”]“ | Stratified meta-belief type +—

E.1.2 Expression Grammar

The expression grammar defines the syntactic forms of CLAIR programs.

+— +Category | Production | Description Variables | e ::= x | Variable reference Lambdas | e ::= “λ”x“:”A“.”e | Anonymous function Application | e ::= e e | Function application Pairs | e ::= “(“e “, “ e”)” | Ordered pair Projections | e ::= e “.” “1” | First projection | | e ::= e “.” “2” | Second projection Injections | e ::= “inl@”B “(“e”)” | Left injection | | e ::= “inr@”A “(“e”)” | Right injection Case | e ::= “case” e “of” “inl” x “=>” e “|” “inr” y “=>” e | Sum elimination Literals | e ::= n | Natural number literal | | e ::= “true” | “false” | Boolean literals | | e ::= ““"s”"” | String literal | | e ::= “()” | Unit literal Beliefs | e ::= “belief(“e “, “ c “, “ j”)” | Belief constructor | | e ::= “val(“e”)” | Extract belief value | | e ::= “conf(“e”)” | Extract belief confidence | | e ::= “just(“e”)” | Extract belief justification Derivation | e ::= “derive(“e “, “ e”)” | Combine beliefs (conjunctive) Aggregation | e ::= “aggregate(“e “, “ e”)” | Aggregate beliefs (independent) Defeat | e ::= “undercut(“e “, “ e”)” | Apply undercut | | e ::= “rebut(“e “, “ e”)” | Apply rebuttal Introspection | e ::= “introspect(“e”)” | Safe self-reference Let Binding | e ::= “let” x “=” e “in” e | Local binding +—

E.1.3 Abstract Syntax

The abstract syntax uses de Bruijn indices for variable representation.

Definition The abstract syntax “Expr” is defined inductively with the following constructors:

“var(n)” — Variable at de Bruijn index n “lam(A, e)” — Lambda abstraction “λ”:“A”.“e “app(e₁, e₂)” — Function application e₁ e₂ “pair(e₁, e₂)” — Ordered pair “fst(e)” — First projection “snd(e)” — Second projection “inl(B, e)” — Left injection “inr(A, e)” — Right injection “case(e, e₁, e₂)” — Case analysis “litNat(n)” — Natural literal “litBool(b)” — Boolean literal “litString(s)” — String literal “litUnit” — Unit literal “belief(v, c, j)” — Belief constructor “val(e)” — Value extraction “conf(e)” — Confidence extraction “just(e)” — Justification extraction “derive(e₁, e₂)” — Derivation “aggregate(e₁, e₂)” — Aggregation “undercut(e, d)” — Undercut “rebut(e₁, e₂)” — Rebuttal “introspect(e)” — Introspection “letIn(e₁, e₂)” — Let binding

The “Justification” type tracks derivation structure: “axiomJ(name)” — Named axiom “rule(name, js)” — Named rule with premises “agg(js)” — Aggregation “undercut_j(j, d)” — Undercut application “rebut_j(j₁, j₂)” — Rebuttal application

E.1.4 Well-Formedness

A type is well-formed if all confidence bounds are in [0,1].

Definition The judgment “wf(A)” holds when:

“wf(B)” for all base types B “wf(A)” ∧ “wf(B)” ⟹ “wf(A → B)” “wf(A)” ∧ “wf(B)” ⟹ “wf(A × B)” “wf(A)” ∧ “wf(B)” ⟹ “wf(A + B)” “wf(A)” ∧ c ∈ [0,1] ⟹ “wf(Belief<”A“[“c”])” “wf(A)” ∧ c ∈ [0,1] ⟹ “wf(Meta<”A“[“n”]”[“c”])“

E.2 Static Semantics (Type System)

E.2.1 Typing Contexts

A typing context Γ maps variable indices to type-confidence pairs.

Γ ::= “∅” | Γ “, “ ⟨A, c⟩

Definition The lookup operation Γ.lookup(n) returns the type-confidence pair at index n (0-indexed from most recent binding).

E.2.2 Typing Judgment Form

The primary typing judgment has the form:

Γ “⊢” e “:” A “@” c

“In context Γ, expression e has type A with confidence bound c.”

E.2.3 Typing Rules

The following rules define well-typed CLAIR expressions. Confidence propagation is explicit in each rule.

T-Variable: Γ “⊢” “var(n)” “:” A “@” c if Γ.lookup(n) = ⟨A, c⟩

T-Abstraction: If Γ “, “ ⟨A, c_A⟩ “⊢” e “:” B “@” c_B, then Γ “⊢” “lam(A, e)” “:” (A “→” B) “@” c_B

T-Application: If Γ “⊢” e₁ “:” (A “→” B) “@” c₁ and Γ “⊢” e₂ “:” A “@” c₂, then Γ “⊢” “app(e₁, e₂)” “:” B “@” (c₁ “×” c₂)

Confidence multiplies for conjunctive derivation.

T-Pair: If Γ “⊢” e₁ “:” A “@” c₁ and Γ “⊢” e₂ “:” B “@” c₂, then Γ “⊢” “pair(e₁, e₂)” “:” (A “×” B) “@” (c₁ “×” c₂)

T-Fst/T-Snd preserve confidence.

T-Inl/T-Inr preserve confidence.

T-Case: If Γ “⊢” e “:” (A “+” B) “@” c and Γ “, “ ⟨A, c⟩ “⊢” e₁ “:” C “@” c₁ and Γ “, “ ⟨B, c⟩ “⊢” e₂ “:” C “@” c₂, then Γ “⊢” “case(e, e₁, e₂)” “:” C “@” (c “⊕” c₁ “⊕” c₂)

Uses probabilistic OR for confidence aggregation.

T-Belief: If Γ “⊢” v “:” A “@” 1 and c_bound “≤” c_actual, then Γ “⊢” “belief(v, c_actual, j)” “:” “Belief<”A“[“c_bound”]” “@” c_bound

The belief's actual confidence c_actual must meet or exceed the declared bound c_bound.

T-Val/T-Conf/T-Just preserve the enclosing confidence.

T-Derive: If Γ "⊢" e1 ":" "Belief<"A"["c1"]" "@" c_e1 and Γ "⊢" e2 ":" "Belief<"B"["c2"]" "@" c_e2, then Γ "⊢" "derive(e1, e2)" ":" "Belief<"A "×" B"["c1 "×" c2"]" "@" (c_e1 "×" c_e2)

Confidence multiplies: both premises must be true.

T-Aggregate: If Γ "⊢" e1 ":" "Belief<"A"["c1"]" "@" c_e1 and Γ "⊢" e2 ":" "Belief<"A"["c2"]" "@" c_e2, then Γ "⊢" "aggregate(e1, e2)" ":" "Belief<"A"["c1 "⊕" c2"]" "@" (c_e1 "⊕" c_e2)

Uses probabilistic OR: independent evidence.

T-Undercut: If Γ "⊢" e ":" "Belief<"A"["c"]" "@" c_e and Γ "⊢" d ":" "Belief["d_c"]" "@" c_d, then Γ "⊢" "undercut(e, d)" ":" "Belief<"A"["undercut(c, d_c)"]" "@" (c_e "×" c_d)

where "undercut(c, d)" = c "×" (1 "−" d).

T-Rebut: If Γ "⊢" e_for ":" "Belief<"A"["c_for"]" "@" c_e1 and Γ "⊢" e_against ":" "Belief<"A"["c_against"]" "@" c_e2, then Γ "⊢" "rebut(e_for, e_against)" ":" "Belief<"A"["rebut(c_for, c_against)"]" "@" (c_e1 "×" c_e2)

where "rebut(c_for, c_against)" = c_for "

T-Introspect: If Γ "⊢" e ":" "Meta<"A"["m"]"["c"]" "@" c_e and m "<" n, then:

The resulting type is: "Meta<"Meta<"A"["m"]"["c"]"">["n"]"["g(c)"]" (a meta-belief about a meta-belief).

Thus: Γ "⊢" "introspect(e)" ":" "Meta<"Meta<"A"["m"]"["c"]"">["n"]"["g(c)"]" "@" c_e

where "g(c)" = c^2 is the Löb discount function (to prevent bootstrapping).

Requires level constraint: only higher levels can introspect lower levels.

E.2.4 Subtyping

CLAIR supports subtyping based on confidence bounds.

S-Belief: "Belief<"A"["c"]" "<:" "Belief<"A"["c']"" if c "≥" c'

Higher confidence is a subtype of lower confidence.

S-Meta follows the same rule.

T-Sub: If Γ "⊢" e ":" A "@" c and A "<:" A' and c "≥" c', then Γ "⊢" e ":" A' "@" c'

Allows weakening both type and confidence.

E.3 Dynamic Semantics

E.3.1 Values

A value is a fully evaluated expression.

> **Definition** The predicate "IsValue(e)" holds for:
> **All lambda abstractions "lam(A, e)" All pairs "pair(v1, v2)" where v1, v2 are values All injections "inl(B, v)" and "inr(A, v)" where v is a value All literals ("litNat", "litBool", "litString", "litUnit") All belief constructors "belief(v, c, j)" where v is a value**

E.3.2 Small-Step Operational Semantics

The small-step relation e "→" e' defines single-step reduction using call-by-value evaluation order.

Beta Reduction: "(λ":"A"." e) v" "→" "e[x := v]" if "IsValue(v)"

Context Rules: "app(e1, e2)" "→" "app(e1′, e2)" if e1 "→" e1′ "app(v1, e2)" "→" "app(v1, e2′)" if e2 "→" e2′ and "IsValue(v1)"

Projections: "fst(pair(v1, v2))" "→" v1 if "IsValue(v1)", "IsValue(v2)" "snd(pair(v1, v2))" "→" v2 if "IsValue(v1)", "IsValue(v2)"

Context rules for "fst" and "snd" evaluate subexpressions first.

Case Analysis: When "case(inl(v), e1, e2)" evaluates and v is a value, it reduces to "e1[x := v]". When "case(inr(v), e1, e2)" evaluates and v is a value, it reduces to "e2[y := v]".

Context rule for “case” evaluates the scrutinee first.

Context rule for “case” evaluates the scrutinee first.

Let Binding: “letIn(v, e)” “⟶” “e[x := v]” if “IsValue(v)”

Context rule for “letIn” evaluates the binding first.

Belief Projections: “val(belief(v, c, j))” “⟶” v if “IsValue(v)” “conf(belief(v, c, j))” “⟶” “belief(v, c, j)” if “IsValue(v)” “just(belief(v, c, j))” “⟶” “litString(toString(j))” if “IsValue(v)”

Context rules evaluate subexpressions to values first.

Derivation, Aggregation, Defeat:

These operations evaluate to values using the confidence operations defined in the typing rules. The small-step semantics provides context rules that evaluate both subexpressions to values before computing the result.

For example, when “derive(v1, v2)” has both subexpressions as values, the evaluator computes a new belief with: **Value: “pair(v1.value, v2.value)” Confidence: “v1.confidence × v2.confidence” Justification: “rule("derive", [v1.just, v2.just])”**

Introspection: “introspect(v)” “⟶” v if “IsValue(v)”

Context rule evaluates subexpression first.

E.3.3 Multi-Step Reduction

The multi-step reduction relation e “⟶⟶” e’ is the reflexive-transitive closure of “⟶”.

e “⟶⟶” e if e = e’ e “⟶⟶” e‘‘ if e “⟶” e’ and e’ “⟶⟶” e‘‘

E.3.4 Evaluation Function

The evaluation function “eval(e)” returns the result of reducing e to a value, or fails if:

1. The expression gets stuck (no applicable rule)
2. The expression exceeds the fuel bound (default: 1000 steps)

> **Definition** “eval(e)” = “some(v)” if e “⟶⟶” v and “IsValue(v)” “eval(e)” = “none” otherwise (stuck or out of fuel)

E.4 Well-Formedness Constraints

E.4.1 Acyclicity of Justification Graphs

For deterministic evaluation in the DAG semantics, justification graphs must be acyclic.

> **Definition** A justification graph G = “(V, E)” is **well-formed** iff:
>
> 1. For all nodes v ∈ V, the transitive closure of E starting from v contains no cycles.
> 2. Equivalently: there is no path v “⟶⟶” v for any v ∈ V.
>
> **Enforcement:** In the reference interpreter, this is checked during type checking by tracking provenance and ensuring no belief can transitively depend on itself.

E.4.2 Stratification Constraints

For safe introspection, meta-belief levels must form a strict hierarchy.

> **Definition** The **stratification constraint** requires:
>
> 1. Every “introspect(e)” operation has proof m “<” n where: **m is the level of the source belief n is the level of the resulting meta-belief**
> 2. This is enforced at compile time: the type checker requires a proof term of type m “<” n.
>
> **Consequence:** No belief can introspect itself or any belief that transitively introspects it.

E.4.3 Confidence Bounds

All confidence values must satisfy 0 “≤” c “≤” 1.

Enforcement: The type system checks this at all belief construction points.

E.5 Example Programs

Basic Belief Basic Belief:

```
belief(42, 0.9, axiomJ("sensor-reading"))
```

A belief in the value 42 with confidence 0.9, traced to a sensor reading axiom.

Conjunctive Derivation Derivation:

```
let p = belief("it is raining", 0.8, axiomJ("weather-report")) in
let q = belief("I have an umbrella", 0.9, axiomJ("visual-check")) in
derive(p, q)
```

Combines two beliefs by multiplication, yielding confidence 0.72.

Aggregation Independent Evidence Aggregation:

```
let e1 = belief("stock will rise", 0.6, axiomJ("analyst-a")) in
let e2 = belief("stock will rise", 0.7, axiomJ("analyst-b")) in
aggregate(e1, e2)
```

Uses probabilistic OR: $0.6 \oplus 0.7 = 0.6 + 0.7 - 0.6 \times 0.7 = 0.88$.

Undercut Defeat:

```
let claim = belief("system is secure", 0.95, axiomJ("vendor-claim")) in
let vuln = belief("critical CVE found", 0.8, axiomJ("security-audit")) in
undercut(claim, vuln)
```

Applies undercut: $0.95 \times (1 - 0.8) = 0.19$.

E.6 Summary

This specification provides:

1. **Complete type grammar:** Base types, functions, products, sums, beliefs, and meta-beliefs
2. **Complete expression grammar:** All CLAIR syntactic forms
3. **Static semantics:** 20 typing rules covering all constructs
4. **Dynamic semantics:** Small-step operational semantics with call-by-value evaluation
5. **Well-formedness constraints:** Acyclicity, stratification, and confidence bounds

The specification is sufficient for implementing a conforming CLAIR interpreter and type checker. The Lean 4 formalization in Appendix A provides a machine-checked version of this specification.