[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

The Confidence Algebra

CLAIR's confidence operations form three distinct commutative monoids…

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.

Justification

DAG Structure

Justification requires directed acyclic graphs with labeled edges…

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.

Self-Reference

CPL: Confidence-Bounded Provability Logic

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.

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

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.

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`

| ∀ c_1 c_2, 0 ≤ rebut(c_1,c_2) ≤ 1 |

`Confidence.Rebut`

| ✓

`rebut_comm`

| ∀ c_1 c_2, rebut(c_1,c_2) = rebut(c_2,c_1) |

`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`

| ∀ c_1 c_2, rebut(c_1,c_2) + rebut(c_2,c_1) = 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:

`c₁ × c₂`

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`

| +—+—+—+

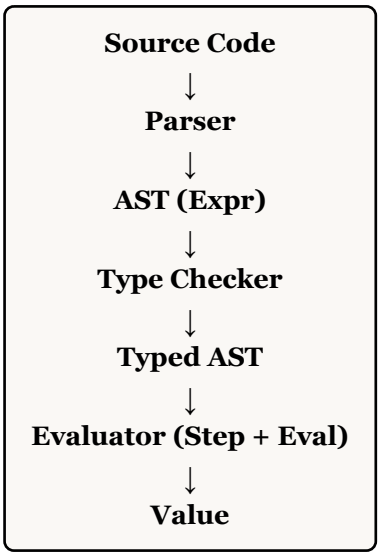

Figure 1: B.1: CLAIR interpreter architecture pipeline

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 $\prec$ be a topological order on V, meaning that if (u, v) ∈ E, then u $\prec$ 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) ⊕ $\bigoplus$ (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 $c_1 \times c_2$. Tracks justification through rule application. | 4 +Aggregation | Combining independent evidence using ⊕: **aggregate(b_1, b_2)** produces belief with confidence $c_1 \oplus c_2$. 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: $\Box(\Box\varphi \to \varphi) \to \Box\varphi$. In provability logic, enables self-reference; in CLAIR, motivates anti-bootstrapping constraint. | 5 +Graded Modality | Modal operators with quantitative gradess. CLAIR's $\Box_{\geq c}$ means "provable with confidence at least **c**." | 5 +Anti-bootstrapping | Principle that self-validating claims cannot increase confidence. Formally: $\mathbf{c} \leq \mathbf{g(c)}$ for some discount function **g**. CLAIR uses $\mathbf{g(c) = c^2}$. | 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 $\mathbf{e \to e'}$ defining one step of computation. Composed to form multi-step evaluation $\mathbf{e \to e'}$. | 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 $\mathbf{undercut(c,d) = c \times (1-d)}$. | 4 +Rebuttal | An argument providing conflicting evidence for the opposite conclusion. Normalizes via $\mathbf{rebut(c_1,c_2) = c_1/(c_1+c_2)}$. | 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 +$\oplus$ | Probabilistic OR: $\mathbf{a \oplus b = a + b - ab}$ | Binary operation | 3 +$\multimap$ | Undercut: $\mathbf{c \multimap d = c \times (1-d)}$ | Binary operation | 4 +× | Multiplication (conjunctive combination) | Binary operation | 3 +**g(c)** | Löb discount function; CLAIR uses $\mathbf{c^2}$ | Unary function | 5 +$\Box_{\geq c}\,\varphi$ | Necessarily φ with confidence at least c | Modal operator | 5 +$\Diamond_{\geq c}\,\varphi$ | Possibly φ with confidence at least c | Modal operator | 5 +$\vdash$ | Typing judgment: $\Gamma \vdash e : A \,@\, c$ | Relation | 10 +$\to$ | Small-step reduc-

tion: 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(c₁, c₂)** | **c₁ / (c₁ + c₂)** | 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(e_1, e_2)” — Function application e_1 e_2 “pair(e_1, e_2)” — Ordered pair “fst(e)” — First projection “snd(e)” — Second projection “inl(B, e)” — Left injection “inr(A, e)” — Right injection “case(e, e_1, e_2)” — 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(e_1, e_2)” — Derivation “aggregate(e_1, e_2)” — Aggregation “undercut(e, d)” — Undercut “rebut(e_1, e_2)” — Rebuttal “introspect(e)” — Introspection “letIn(e_1, e_2)” — 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(j_1, j_2)” — 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 "⊢" e_1 ":" (A "→" B) "@" c_1 and Γ "⊢" e_2 ":" A "@" c_2, then Γ "⊢" "app(e_1, e_2)" ":" B "@" (c_1 "×" c_2)

Confidence multiplies for conjunctive derivation.

T-Pair: If Γ "⊢" e_1 ":" A "@" c_1 and Γ "⊢" e_2 ":" B "@" c_2, then Γ "⊢" "pair(e_1, e_2)" ":" (A "×" B) "@" (c_1 "×" c_2)

T-Fst/T-Snd preserve confidence.

T-Inl/T-Inr preserve confidence.

T-Case: If Γ "⊢" e ":" (A "+" B) "@" c and Γ ", " 〈A, c〉 "⊢" e_1 ":" C "@" c_1 and Γ ", " 〈B, c〉 "⊢" e_2 ":" C "@" c_2, then Γ "⊢" "case(e, e_1, e_2)" ":" C "@" (c "⊕" c_1 "⊕" c_2)

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 "⊢" e_1 ":" "Belief<"A"["c_1"]" "@" c_{e_1} and Γ "⊢" e_2 ":" "Belief<"B"["c_2"]" "@" c_{e_2}, then Γ "⊢" "derive(e_1, e_2)" ":" "Belief<"A "×" B"["c_1 "×" c_2"]" "@" (c_{e_1} "×" c_{e_2})

Confidence multiplies: both premises must be true.

T-Aggregate: If Γ "⊢" e_1 ":" "Belief<"A"["c_1"]" "@" c_{e_1} and Γ "⊢" e_2 ":" "Belief<"A"["c_2"]" "@" c_{e_2}, then Γ "⊢" "aggregate(e_1, e_2)" ":" "Belief<"A"["c_1 "⊕" c_2"]" "@" (c_{e_1} "⊕" c_{e_2})

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_{e_1} and Γ "⊢" e_against ":" "Belief<"A"["c_against"]" "@" c_{e_2}, then Γ "⊢" "rebut(e_for, e_against)" ":" "Belief<"A"["rebut(c_for, c_against)"]" "@" (c_{e_1} "×" c_{e_2})

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 ⊕ 0.7 = 0.6 + 0.7 − 0.6×0.7 = 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 × (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.