A Lean-Verified Clinical Hypergraph Universe: Types, Semantics, and System Invariants

Jack & Aly

February 17, 2026

Abstract

This document specifies a verified clinical decision support kernel built on a hypergraph ruleset. The kernel has four layers: *types* (what exists), *derivation* (how verdicts follow from facts), *invariants* (what states are forbidden), and *authoring contract* (how new clinical knowledge enters safely). Derivation uses *specificity ordering*: when a general rule and a specific exception both fire, the more specific rule takes priority. Lean proves three safety invariants at build time over all possible fact combinations. Conflicts that specificity cannot resolve are surfaced to clinicians as structured clinical questions. The runtime verifier is deterministic, non-probabilistic, and non-optimizing.

1 Overview

The system provides a mathematical guarantee: given a set of patient facts and a proposed clinical action, is that action justified by the ruleset? It does not diagnose, rank options, or reason probabilistically. Clinical soundness of individual rules is a human responsibility; the system guarantees only that the ruleset is internally coherent.

1.1 The components

The document mirrors the four layers of the Lean library:

1. **Types** (Section 3) define the shapes of data: what kinds of objects exist.
2. **Derivation** (Section 4) gives those types operational meaning: how rules fire, how specificity resolves conflicts, and how verdicts are collected.
3. **Invariants** (Section 5) prove guarantees: the ruleset cannot produce forbidden states.
4. **Authoring contract** (Section 6) defines how new knowledge enters the system safely: specificity handles the common case; structured clinical questions handle the rest.

1.2 Two-phase architecture

Build time (Lean). The clinical ruleset is treated as data (a finite set of hyperedges). Lean proves that the ruleset satisfies three safety invariants (Section 5) over the specificity-aware derivation (Section 4). If a new rule introduces a violation, the build fails.

Runtime (verifier). An upstream ontology normalizes patient data into a finite set of canonical fact tokens. Given this fact set and a proposed action, the verifier:

1. matches hyperedge premises by set inclusion,
2. applies specificity ordering to resolve shadowed rules,
3. derives verdicts per the definitions in Section 4,
4. returns *accept* iff the proposal is licensed by the verified ruleset.

1.3 System boundaries

Upstream: ontology. The ontology deterministically produces a complete, transitively expanded set of canonical facts (synonym normalization, subclass closure, diagnosis-attribute expansion, monotonic discretization of continuous values). The verifier receives only the materialized fact set.

Downstream: utility (optional). The verifier may output multiple compatible actions. A downstream, untrusted layer may rank these already-verified outputs (e.g., resource-aware triage, UI prioritization). This is the appropriate place to address decision fatigue. It does not change the formal meaning of the ruleset.

Trusted kernel. The trusted kernel consists of: the definition of Derive (including specificity ordering), the action algebra interfaces (Incompatible, Feasible), and the build-time proofs of the three invariants. Everything else—ontology, UI, conflict resolution interface, downstream ranking—is untrusted.

2 Running Clinical Examples

All examples are illustrative placeholders, not clinical advice. Examples D through F illustrate the authoring contract (Section 6) and specificity ordering (Section 4.2); readers may revisit them after those sections.

2.1 Example A: Severe preeclampsia at ≥ 34 weeks

Ontology output (canonical facts).

`DxAttr.Preeclampsia.Severe`, `Dx.Preeclampsia`, `Dx.HypertensiveDisorder`,
`Ctx.GA_≥28w`, `Ctx.GA_≥34w`

Rule. {`Preeclampsia.Severe`, `GA_≥34w`} → `Obligated(Deliver)`

Runtime. All premises are present. The verifier derives `Obligated(Deliver)` and can certify a proposal to deliver.

2.2 Example B: Harmless co-firing

R1: {`Preeclampsia.Severe`, `GA_≥34w`} → `Obligated(Deliver)`
R2: {`FetalStatus_Reassuring`} → `Allowed(Wait)`

Both may fire simultaneously. This is not an error: their verdicts target different actions and do not contradict.

2.3 Example C: Invariant violation

R1: {`Preeclampsia.Severe`, `GA_≥34w`} → `Obligated(Deliver)`
R2: {`Placenta.Previa`} → `Rejected(Deliver)`

These rules have independent premises: neither premise set is a subset of the other. A patient with severe preeclampsia at ≥ 34 weeks *and* placenta previa would satisfy both premise sets, producing `Obligated(Deliver)` and `Rejected(Deliver)`—a contradiction.

Because the premises are independent, specificity ordering cannot resolve this conflict. Neither rule shadows the other. The ruleset is rejected at build time.

No actual patient is needed. Lean constructs the hypothetical fact set

{ `Preeclampsia.Severe`, `GA_≥34w`, `Placenta.Previa` }

as a proof witness and shows that a contradiction is possible.

Examples D and E show how the authoring contract (Section 6) resolves this by producing a new rule whose premises are a superset of both, introducing a specificity relationship that shadows the contradiction.

2.4 Example D: Specificity resolution

Suppose R1 exists and a clinician tries to add R2:

R1: {`Preeclampsia.Severe`, `GA_≥34w`} → `Obligated(Deliver)`
R2: {`Placenta.Previa`} → `Rejected(Deliver)`

The build fails: the premises are independent, and the verdicts contradict on action `Deliver`. The system *holds* R2 (it is not permanently rejected) and surfaces the conflict as a clinical question (see Example E).

Once the clinician resolves the conflict, the resolution is encoded as R3:

R3: {`Preeclampsia.Severe`, `GA_≥34w`, `Placenta.Previa`} → `Obligated(Deliver.Cesarean)`

R2 and R3 are added together. The build re-runs over the complete ruleset {R1, R2, R3}.

R3's premises are a strict superset of both R1's and R2's premises. For any patient with all three facts, all three rules fire, but R3 *shadows* R1 and R2 via specificity ordering. Only R3's verdict reaches the derived verdict set. No contradiction arises.

R1 continues to apply for patients with severe preeclampsia at ≥ 34 weeks *without* placenta previa. R2 continues to apply for patients with placenta previa *without* severe preeclampsia at ≥ 34 weeks. R3 handles the overlap.

2.5 Example E: Guided conflict resolution

When the build holds R2 (as in Example D), the system surfaces the conflict as a structured clinical question:

Scenario: A patient has severe preeclampsia at ≥ 34 weeks **and** placenta previa.
Rule 1 says: delivery is required.
Rule 2 says: delivery is ruled out.

What should happen for this patient?
(a) Deliver this patient (obligation applies despite placenta previa).
(b) Do not deliver this patient (placenta previa takes priority).
(c) Escalate to MFM specialist (neither rule alone is sufficient).

These three options are always available because they are derived mechanically from the conflict structure (Section 6.4). If the action ontology supports subtypes, the system may also suggest refined actions (e.g., `Deliver.Cesarean`) as additional options—but the base three do not depend on ontology knowledge.

The clinician's answer is encoded as a new rule with premises $P_1 \cup P_2$ and the chosen verdict. Both the held rule and the resolution rule enter the ruleset together. The build re-runs and passes.

2.6 Example F: Genuine clinical uncertainty

When the clinician's answer to a surfaced conflict is "this depends on the specific patient," they select option (c). The system encodes escalation as an action:

```
{Preeclampsia.Severe, GA_≥34w, ExtremePrematurity}
→ Obligated(Escalate.MFM)
```

The conflict resolves by acknowledging that the combination exceeds guideline coverage and routing the case to a specialist. Escalation actions are first-class members of the Action type and participate in all invariant checks. Genuine clinical uncertainty does not break the model; it means the action is escalation rather than treatment.

3 Core Types

Types define what can be named, not what is true or what happens.

3.1 Facts

An opaque atomic token from the ontology: $\mathsf{Fact} : \mathsf{Type}$.

3.2 Actions

A canonical token representing a clinical intervention, plan step, or escalation: $\mathsf{Action} : \mathsf{Type}$.

Treatment actions (e.g., `Deliver`, `Administer(MgSO4)`) and escalation actions (e.g., `Escalate.MFM`, `Consult.Specialist`) share the same type.

3.3 Verdicts

A fixed vocabulary of judgments about actions:

$$\mathsf{Verdict} \in \{\, \mathsf{Obligated},\ \mathsf{Allowed},\ \mathsf{Disallowed},\ \mathsf{Rejected} \,\}.$$

- $\mathsf{Obligated}(a)$: action a must be taken.
- $\mathsf{Allowed}(a)$: action a may be taken.
- $\mathsf{Disallowed}(a)$: action a is not licensed by the ruleset. This is the default—actions are disallowed unless explicitly allowed or obligated.
- $\mathsf{Rejected}(a)$: action a is actively ruled out (e.g., contraindicated).

3.4 Conflicting verdict pairs

Two verdicts on the same action are *conflicting* iff they form one of the following pairs:

- $\mathsf{Obligated}(a)$ and $\mathsf{Rejected}(a)$,
- $\mathsf{Allowed}(a)$ and $\mathsf{Rejected}(a)$.

All other verdict combinations on the same action are non-conflicting and may coexist. This definition is referenced by the specificity ordering (Section 4.2) and the invariants (Section 5).

3.5 Rules (hyperedges)

A rule pairs a finite set of premises with a verdict:

$$\mathsf{Rule} \equiv \big(\mathsf{premises} : \mathcal{P}_f(\mathsf{Fact})\big) \times \big(\mathsf{out} : \mathsf{Verdict}(\mathsf{Action})\big).$$

Written as: $\{p_1, \ldots, p_k\} \to v(a)$. Each rule maps one premise set to one verdict on one action.

3.6 Action algebra interfaces

Two predicates capture clinical relationships between actions. Their signatures are declared in the Lean library; their concrete content comes from versioned data artifacts.

Incompatibility. $\mathsf{Incompatible} : \mathsf{Action} \to \mathsf{Action} \to \mathsf{Prop}.$
Actions a and b cannot both be executed in the same plan instance.

Feasibility. $\mathsf{Feasible} : \mathsf{Action} \to \mathcal{P}_f(\mathsf{Fact}) \to \mathsf{Prop}.$
Action a is implementable given facts F.

4 Derivation

This section defines the operational semantics of the verifier. These are definitions in the Lean sense: they give the types their meaning.

4.1 Rule firing

A rule $r = (P \to v(a))$ *fires* iff $P \subseteq F$. Firing is purely set-based.

4.2 Specificity ordering

When multiple fired rules target the same action with conflicting verdicts (as defined in Section 3.4), the most specific rule takes priority.

A rule r_2 is *more specific* than r_1 (written $r_1 \prec r_2$) iff $P_{r_1} \subset P_{r_2}$ (strict subset of premises).

A fired rule r_1 targeting action a is *shadowed* if there exists a fired rule r_2 such that:

1. r_2 also fires ($P_{r_2} \subseteq F$),
2. r_2 targets the same action a,
3. $r_1 \prec r_2$ (r_2 has a strict superset of r_1's premises),

4. r_1 and r_2 have conflicting verdicts on action a.

Shadowed rules do not contribute to the derived verdict set.

Specificity can only resolve conflicts where one premise set is a subset of the other. When premises are independent (neither is a subset of the other), specificity does not apply and the conflict must be resolved through the authoring contract (Section 6).

4.3 Deriving verdicts

Let R be the ruleset and F the runtime fact set.

$$\begin{aligned}
\mathsf{Fired}(R,F) &= \{\, r \in R \mid P_r \subseteq F \,\} \\
\mathsf{Shadowed}(R,F) &= \big\{\, r_1 \in \mathsf{Fired}(R,F) \;\big|\; \exists\, r_2 \in \mathsf{Fired}(R,F) \text{ s.t.} \\
&\qquad \mathsf{sameAction}(r_1,r_2) \;\wedge\; P_{r_1} \subset P_{r_2} \;\wedge\; \mathsf{conflicting}(r_1,r_2) \,\big\} \\
\mathsf{Derive}(R,F) &= \big\{\, v(a) \;\big|\; (P \to v(a)) \in \mathsf{Fired}(R,F) \setminus \mathsf{Shadowed}(R,F) \,\big\}
\end{aligned}$$

Where $\mathsf{conflicting}(r_1, r_2)$ holds iff r_1 and r_2 produce a conflicting verdict pair on the same action (Section 3.4).

Two properties hold by construction:

- **Determinism.** $\mathsf{Derive}(R, F)$ is a pure function of (R, F).
- **Specificity soundness.** A verdict appears in Derive only if its rule fires and is not shadowed by a more specific rule with a conflicting verdict on the same action.

4.4 Certificate checking

Given a proposed action $a^\star$:

- **Accept** if $\mathsf{Obligated}(a^\star) \in \mathsf{Derive}(R, F)$, or $\mathsf{Allowed}(a^\star) \in \mathsf{Derive}(R, F)$ with no rejection.
- **Reject** otherwise.

4.5 Complexity

Premise matching is $O(|R| \cdot \overline{|P|})$.
With a fact-to-rule index, candidate sets are small and runtime cost is effectively constant per case.

5 System Invariants

Lean proves that ruleset R satisfies three properties over the specificity-aware Derive. These are proven once per ruleset version at build time. If any fails, the version is rejected.

5.1 No contradictory verdicts

For all F and a, letting $D = \mathsf{Derive}(R, F)$:

$$\neg\big(\mathsf{Obligated}(a) \in D \;\wedge\; \mathsf{Rejected}(a) \in D\big) \tag{1}$$

$$\neg\big(\mathsf{Allowed}(a) \in D \;\wedge\; \mathsf{Rejected}(a) \in D\big) \tag{2}$$

5.2 No incompatible obligations

For all F:

$$\neg\exists\, a, b, \quad \mathsf{Obligated}(a) \in D \;\wedge\; \mathsf{Obligated}(b) \in D \;\wedge\; \mathsf{Incompatible}(a, b) \tag{3}$$

5.3 Ought implies can

For all F and a:

$$\mathsf{Obligated}(a) \in D \;\Rightarrow\; \mathsf{Feasible}(a, F) \tag{4}$$

5.4 Role of specificity in invariant checking

Specificity ordering reduces the burden on invariant proofs by eliminating most contradictions before they reach the checker. Two rules with a subset relationship on premises are resolved by specificity within Derive; Lean only needs to verify the invariants for rule pairs with independent premises (neither is a subset of the other).

6 Authoring Contract

The authoring contract defines how new clinical knowledge enters the system while preserving coherence at every step. This is a core layer of the system, not an implementation convenience.

6.1 Principle

Clinicians write simple positive rules that mirror clinical guidelines. No negation tokens, guard conditions, or awareness of other rules is required. The clinician expresses what they know; the system determines whether it is globally consistent.

6.2 Three entry paths

When a clinician submits a new rule, exactly one of three things happens:

Path 1: Clean entry. The new rule does not conflict with any existing rule under any possible fact set. The build passes. The rule enters the ruleset directly.

Path 2: Specificity resolution. The new rule conflicts with an existing rule, but a more specific rule already exists (or the new rule itself is more specific) whose premises are a superset of both conflicting rules. Specificity ordering within Derive shadows the less specific rule. The build passes without human intervention.

Path 3: Guided resolution. The new rule conflicts with an existing rule, the premises are independent (neither is a subset of the other), and no existing specific rule resolves the overlap. The build fails. The new rule is *held*—not permanently rejected, but not yet added to the shipped ruleset. The system surfaces the conflict as a clinical question (Section 6.3). Once the clinician answers, the held rule and the resolution rule enter the ruleset together. The build re-runs over the complete ruleset and must pass all three invariants.

6.3 Conflict surfacing

When Path 3 is triggered, the system constructs the hypothetical fact set $P_1 \cup P_2$ (the union of both rules' premises) and presents the conflict as a structured clinical question:

> **Scenario:** A patient has [premises of rule 1] **and** [premises of rule 2].
> Rule X says: [verdict 1]. Rule Y says: [verdict 2].
>
> **What should happen for this patient?**
> (a) [Verdict 1 applies to this combination]
> (b) [Verdict 2 applies to this combination]
> (c) Escalate to specialist

6.4 Option generation

The three base options are derived mechanically from the conflict structure alone, with no clinical knowledge or ontology lookup required. Every conflict has the same shape: Rule A says verdict v_1 on action a, Rule B says verdict v_2 on action a, and v_1 and v_2 are a conflicting pair (Section 3.4).

- **Option (a):** Rule A's verdict applies to the overlap. Encoded as: $P_A \cup P_B \to v_1(a)$.
- **Option (b):** Rule B's verdict applies to the overlap. Encoded as: $P_A \cup P_B \to v_2(a)$.
- **Option (c):** Escalate. Encoded as: $P_A \cup P_B \to \mathsf{Obligated}(\mathsf{Escalate}.\mathit{Specialist})$.

These three options are always available. If the action ontology supports subtypes (e.g., `Deliver.Cesarean` as a subtype of `Deliver`), the system may present additional refined options. Such enhanced options depend on ontology knowledge and are not guaranteed to be available.

In all cases, the chosen option produces a rule whose premises are $P_A \cup P_B$—a strict superset of both conflicting rules' premises. This ensures that specificity ordering will shadow the original conflict going forward.

6.5 Encoding genuine uncertainty

When the clinician selects option (c), the system encodes an obligation to escalate. The conflict resolves not by choosing a side but by acknowledging that the combination exceeds guideline coverage and must be routed to a specialist. Genuine clinical uncertainty does not break the model; it means the action is escalation rather than treatment.

6.6 Coherence guarantee

At every point in time, the shipped ruleset passes all three invariants. A conflicting rule is held during resolution and never enters the shipped ruleset until its resolution rule accompanies it and the combined ruleset passes the build. No incoherent ruleset exists in production.

7 Implementation Notes

7.1 Ruleset as versioned data

1. Maintain hyperedges as a versioned data artifact.
2. Maintain $\mathsf{Incompatible}$ and $\mathsf{Feasible}$ as versioned data.

3. In CI/build, load into Lean and run invariant proofs over the specificity-aware Derive.

4. If a proof fails, hold the new rule and surface the conflict via the authoring contract.

5. Once resolved, add the held rule and the resolution rule together, re-run the build, and ship.

7.2 Certificates

An acceptance certificate includes: the proposed action $a^\star$, the fact set hash, the identifiers of fired (unshadowed) rules that justify the verdict, and a ruleset version identifier. The certificate is an auditable runtime trace, not a Lean proof.

7.3 Proof automation

As the ruleset grows, tactic scripts and metaprograms construct proofs of the three invariants more efficiently. Incremental checking means a new rule is checked against existing rules for co-firing violations, not the entire ruleset from scratch. Specificity further reduces the proof burden by resolving subset-related conflicts within Derive, so Lean only needs to verify invariants for rule pairs with independent premises.

A Formal Reference

A.1 Types

- $\mathsf{Fact} : \mathsf{Type}$
- $\mathsf{Action} : \mathsf{Type}$ (includes escalation actions)
- $\mathsf{Verdict} \in \{\mathsf{Obligated}, \mathsf{Allowed}, \mathsf{Disallowed}, \mathsf{Rejected}\}$
- $\mathsf{Rule} \equiv \mathcal{P}_f(\mathsf{Fact}) \times \mathsf{Verdict}(\mathsf{Action})$
- $\mathsf{Incompatible} : \mathsf{Action} \to \mathsf{Action} \to \mathsf{Prop}$
- $\mathsf{Feasible} : \mathsf{Action} \to \mathcal{P}_f(\mathsf{Fact}) \to \mathsf{Prop}$

A.2 Conflicting verdict pairs

Two verdicts on the same action a conflict iff they are:

- $(\mathsf{Obligated}(a), \mathsf{Rejected}(a))$
- $(\mathsf{Allowed}(a), \mathsf{Rejected}(a))$

A.3 Derivation with specificity

$$\begin{aligned}
\mathsf{Fired}(R, F) &= \{\, r \in R \mid P_r \subseteq F \,\} \\
\mathsf{Shadowed}(R, F) &= \big\{\, r_1 \in \mathsf{Fired}(R, F) \;\big|\; \exists\, r_2 \in \mathsf{Fired}(R, F) \text{ s.t.} \\
&\qquad \mathsf{sameAction}(r_1, r_2) \;\wedge\; P_{r_1} \subset P_{r_2} \;\wedge\; \mathsf{conflicting}(r_1, r_2) \,\big\} \\
\mathsf{Derive}(R, F) &= \big\{\, v(a) \;\big|\; (P \to v(a)) \in \mathsf{Fired}(R, F) \setminus \mathsf{Shadowed}(R, F) \,\big\}
\end{aligned}$$

A.4 Invariants

For all F, letting $D = \mathsf{Derive}(R, F)$:

1. $\forall a : \neg(\mathsf{Obligated}(a) \in D \land \mathsf{Rejected}(a) \in D) \quad \text{and} \quad \neg(\mathsf{Allowed}(a) \in D \land \mathsf{Rejected}(a) \in D)$
2. $\neg\exists\, a, b : \mathsf{Obligated}(a) \in D \land \mathsf{Obligated}(b) \in D \land \mathsf{Incompatible}(a, b)$
3. $\forall a : \mathsf{Obligated}(a) \in D \Rightarrow \mathsf{Feasible}(a, F)$