Deterministic reimbursement rules in Lean

Matha Labs | Research prototype v0.1.0 | November 15, 2026

Abstract

This paper describes a small executable reimbursement-rule experiment implemented in Lean 4. A reviewed claim is checked against an invented, versioned payer contract. Accepted inputs produce an integer-cent allocation between a synthetic payer and patient. Thirteen named theorems establish eligibility preconditions, rejection behavior, amount bounds, and conservation of the allowed amount. Twenty-nine synthetic fixtures exercise policy mismatches, missing facts, effective dates, unit bounds, and arithmetic edge cases. The project demonstrates a deterministic checking architecture. It does not implement a real insurer's payment system, estimate actual benefits, or guarantee reimbursement.

1. Why diagnosis information is insufficient

A clinical diagnosis identifies a condition; it does not by itself identify the service, place of service, applicable payer contract, locality, authorization, or the patient's benefit state. Treating a code as a direct lookup key for a payment amount collapses distinct questions. The CMS Physician Fee Schedule materials illustrate this separation: payment information is organized around services and includes locality-related adjustments and policy indicators [1, 2].

Our experiment models the contract-checking stage independently from the clinical-coding stage. Its `diagnosisSupported` field is an externally reviewed Boolean input. No diagnosis string is used as a substitute for a service identifier. A fixture explicitly tests that substituting `I10` for the invented service identifier is refused. This simple boundary makes it possible to discuss formal arithmetic without suggesting that an ICD-10-CM code determines a real claim's payment.

2. Architecture and relation to Lexifina

Lexifina's case-law article describes extracted claims paired with evidence and checked against formal rules [3]. The relevant architectural idea is to constrain the final decision through an independently executable checker. We use source identity and revision fields to bind a claim to the selected synthetic policy. Every result can be reproduced from those inputs and the implementation.

This release does not include a language model, learned weights, model training, retrieval system, natural-language policy compiler, or automatic extraction of insurance facts. A future generative component could propose a structured claim, but its output would remain subject to review and deterministic checking. A complete system would additionally need evidence objects for coverage and authorization, authenticated policy ingestion, and explicit source-document citations. The current structures encode reviewed facts rather than checking their documentary origin.

3. Synthetic policy definition

The invented policy has source `urn:mathalabs:synthetic-payer`, revision `fixture-v1`, payer `SYNTH-PAYER`, service `SYNTH-CONSULT`, and locality `SYNTH-LA`. It applies to office services from January 1, 2026 up to but excluding January 1, 2027. Its fixture allowance is 12,500 cents per unit, its maximum unit count is four, and its fixed copay is 2,500 cents.

These values are authored examples. They are not Medicare amounts, a commercial contract, a CPT or HCPCS descriptor, or an actual Medicare locality. We do not distribute a procedure-code corpus. The fixture interval is selected to make period boundaries testable; it does not represent an actual benefit period. The repository's source manifest states exactly which values are invented and which external publications provide background.

CMS describes the Physician Fee Schedule in terms of work, practice expense, and malpractice relative value units, geographic adjustments, and a conversion factor [2]. Our calculation deliberately uses a different, much smaller contract. It must not be presented as an implementation of that formula or a replacement for definitive payment files and payer adjudication.

4. Reviewed claim model

`Claim` contains policy source and revision, payer, service identifier, locality, place of service, service date, units, billed cents, remaining deductible cents, coverage, authorization, network status, and diagnosis support. Coverage and authorization use `Option Bool`. `none` means the fact is unknown, `some false` means a reviewed negative fact, and `some true` means a reviewed positive fact. Both unknown and negative states prevent a quote.

Dates use natural-number YYYYMMDD values. A calendar validator checks Gregorian month lengths and leap years. The policy start and exclusive end must also be valid dates, and the interval must be ordered correctly. Monetary values and units use natural numbers. This eliminates negative cent amounts at the type level, while eligibility rejects zero or excessive units. There is no floating-point arithmetic or unspecified currency rounding in the implemented contract.

5. Eligibility checker

`Conditions` requires matching policy source, revision, payer, service, and locality; an office place of service; an ordered and applicable date interval; a positive unit count within the contract limit; positive reviewed coverage and authorization; in-network status; supported diagnosis information; and valid calendar dates for the claim and both policy endpoints. `verify` returns the decidable Boolean form of that proposition.

`adjudicate` returns `some Quote` only when verification succeeds. Otherwise it returns `none`. A caller must not convert `none` to zero payment: rejection means the model has not produced an allocation under its explicit conditions. The present result type does not distinguish every rejection reason. A production-oriented extension would return a typed explanation and missing-evidence list, while preserving the same acceptance conditions.

6. Integer-cent allocation

For billed amount B, contract rate R, units U, remaining deductible D, and fixed copay C, the fixture calculates:

- Allowed amount A = min(B, R times U).
- Applied deductible X = min(D, A).
- Residual S = A minus X.
- Applied copay Y = min(C, S).
- Patient amount = X plus Y.
- Payer amount = S minus Y.

Every operation is over natural-number cents. The `min` operations cap both cost-sharing stages before subtraction. In the base fixture B is 20,000, R is 12,500, U is one, D is 1,000, and C is 2,500. The allowed amount is 12,500 cents, the patient allocation is 3,500 cents, and the synthetic payer allocation is 9,000 cents. A very large deductible or copay assigns at most the allowed amount to the patient, and a zero billed amount produces three zero amounts.

7. Proof inventory

The thirteen named theorems appear in `Reimbursement/Model.lean`:

- `verification_iff` connects successful Boolean checking to the explicit eligibility proposition.
- `accepted_requires_coverage`, `accepted_requires_authorization`, and `accepted_requires_supported_diagnosis` establish required reviewed facts.
- `accepted_date_in_window` establishes both effective-date inequalities.
- `unknown_coverage_refused` and `unknown_authorization_refused` establish rejection of missing facts.
- `rejected_has_no_quote` prevents a failed verifier from emitting an allocation through `adjudicate`.
- `allowed_le_billed` and `allowed_le_contract` bound the allowance by both inputs.
- `conservation` proves patient cents plus payer cents equals allowed cents for every policy and claim value.
- `patient_le_allowed` and `payer_le_allowed` prove individual amount bounds.

The conservation theorem is universal over the represented natural-number amounts, rather than limited to the numerical example. It establishes an arithmetic invariant, not a claim about what any real payer owes. The code uses Lean's bundled standard library, pins version 4.35.0, and contains no custom axioms, unsafe declarations, admitted proofs, or proof placeholders.

8. Synthetic evaluation

Twenty-nine fixtures test the reviewed base claim and changes to individual requirements. They include unknown and denied coverage, unknown and denied authorization, payer and service mismatches, locality and setting mismatches, network status, unsupported diagnosis information, zero and excessive units, the maximum unit boundary, policy-source and revision mismatches, and date-boundary errors. Calendar cases include an impossible February date, an invalid policy endpoint, and a valid leap day.

Arithmetic cases cover the expected allocation, a lower billed amount, an oversized deductible, a zero charge, and an oversized copay. All fixtures are evaluated in a compile-time `example` as well as the executable. At publication preparation, the project built with the pinned toolchain and reported 29/29 passing synthetic fixtures. These cases are regression evidence, not claims-processing accuracy, agreement with an insurer, a payment

recovery rate, or an estimate of external validity.

9. Trust boundary and limitations

The checker trusts the selected policy parameters and externally reviewed benefit facts. Matching a policy-source string does not authenticate an insurer, establish an executed contract, or verify that a coverage decision is current. The `diagnosisSupported` flag does not itself prove medical necessity. Those facts require evidence and upstream governance outside this release.

The contract omits coordination of benefits, claim edits, bundling, modifiers, balance billing, deductibles shared across claims, benefit accumulators, retroactive changes, appeals, taxes, currency conversion, sequestration, and payer-specific rounding. It has no transactional state, so it cannot safely decrement a shared deductible across concurrent claims. It also does not model actual deductible and copay interaction rules for any insurer. Determinism guarantees repeatable behavior for fixed inputs; it cannot supply missing policy or patient facts.

10. Next experiments

Useful next steps are a reviewed and licensed policy fixture with explicit evidence citations, an independent reference implementation, and a held-out set of de-identified adjudicated claims where access is authorized. Evaluation should report missing-input and abstention rates separately from arithmetic agreement. Any stateful benefit ledger should specify concurrency and replay semantics before integration. These are proposed research extensions and have not been implemented here.

References

1. CMS. Physician Fee Schedule Look-up Tool Overview. https://www.cms.gov/medicare/physician-fee-schedule/search/overview

2. CMS. Physician Fee Schedule. https://www.cms.gov/cms-guide-medical-technology-companies-and-other-interested-parties/payment/physician-fee-schedule

3. Lexifina. Formalizing all US case law in Lean. November 13, 2026. https://lexifina.com/blog/formalizing-all-us-case-law-in-lean

4. Lean. Installation and language resources. https://lean-lang.org/install/