From Unstructured Regulations to Compliance Agents: Structuring Legal Text for Machine Reasoning

Pradeep Pujari

May 2025

1 Introduction

Current compliance processes rely on manual review of regulatory documents, leading to inconsistent interpretations and delayed decision-making. Also, compliance teams cannot quickly access accurate regulatory guidance from fragmented PDF documents, resulting in inconsistent decisions and regulatory risk. Legal teams spend 40% of their time manually searching through 500+ page regulatory documents to answer basic compliance questions. The compliance landscape is filled with dense PDFs containing critical rules, definitions, and guidelines. Current large language models, while powerful, rely heavily on transformer architecture and auto-regressive processing, which may limit their direct applicability to compliance-related use cases. There are two main information in these PDF files. One is legal definitions and second conditional statements or we call it rules. First, we extract the essential information—rules, definitions, and lemmas—from the existing PDF documents. This isn't just simple text extraction; it's about identifying the crucial elements that define compliance in the field. We then transform this information into structured JSON documents that a machine can effectively process. This extraction step is the foundation that makes everything else possible. It converts dense regulatory text into clear, structured data that can be analyzed and applied consistently. The second step builds upon this foundation to create a specialized compliance language model that can interpret, apply, and explain regulations. We lack an existing LLM model that can be used directly. So, the proposed solution necessitates an application architecture based on an agentic pipeline approach.

2 Solution

The goal involves rules like: "The company must have board approval and shareholder resolution before issuing depository receipts." This is a strongly normative domain, where:

Rules are explicit and prescriptive AND
Violations have clear consequences AND
Compliance is binary (either fulfilled or not)

We propose a three step process.
1. Extracting legal definitions
2. Extracting Rules from PDF documents.
3. Applying to a new document for compliance checking.
Often definitions have complex relationships. A simple key-value symbol table might not capture the hierarchical nature of legal frameworks. To make it simple, we did not consider hierarchical relations in this MVP.

1. Knowledge Graphs

Knowledge graphs offer a powerful alternative that explicitly models relationships between legal concepts: Nodes: Legal terms, concepts, statutes, and provisions Edges: Relationships like "defines," "amends," "references," "supersedes," or "exempts" Properties: Attributes such as effective dates, jurisdictions, and authority This approach allows you to capture important legal relationships like: Term A is defined by Statute B Definition C applies only within context D Section E amends the definition in Section F

2. Hierarchical Document Stores (MongoDB)

```
{
  "Act": {
    "definition": "the Securities Contracts (Regulation) Act, 1956",
    "reference": "42 of 1956",
    "related_terms": {
      "appeal": {
        "definition": "an appeal filed under section 21A...",
        "context": ["Securities Contracts"]
      }
    }
  }
}
```

2. NetworkX for Legal Definitions

NetworkX is a pure Python package for creating, manipulating, and studying complex networks. It's particularly well-suited for legal definitions because: Advantages for Legal Knowledge:

Intuitive API: Simple to create and manipulate graph structures Rich attributes: Nodes and edges can store arbitrary JSON data Built-in algorithms: Offers path finding, centrality measures, and graph traversal Visualization: Integrated with matplotlib for visual analysis Serialization: Easy export/import to various formats including JSON

3. Distributed Representations (Embeddings)

In contrast, distributed representations encode meaning as patterns across many dimensions. Each word/concept is represented as a vector (typically 100-1000

dimensions). Meaning is distributed across all dimensions and similar concepts have similar vector patterns.

Symbol tables handle:

Explicit information storage (definitions, sources, relationships) Structural information (hierarchies, cross-references) Exact matching and lookups

Embeddings handle:

Semantic similarity Contextual understanding Generalization to new examples hybrid system works:

Embedding component:

Converts "appeal," "valid," and "securities regulation" to vectors Finds semantically similar concepts (even if exact terms aren't in the query) Identifies that this is related to the Securities Regulation Act

Symbol table component:

Retrieves precise definition of "appeal" from Rule 2(1)(b) Accesses structural information about where this definition applies Provides exact citation and reference information

Advantages of This Hybrid Approach

Precision with flexibility: Symbol tables provide exact definitions while embeddings capture semantic nuance
Handling of unseen inputs: Embeddings allow the system to understand queries using terminology not explicitly defined
Contextual disambiguation: Embeddings help distinguish different meanings of the same term in different contexts
Enhanced retrieval: Combining exact and similarity-based matching improves information retrieval
Learning capabilities: Embeddings can be fine-tuned to improve over time
Context injection approach: Will include stored definitions as part of the context/prompt when asking the LLM to analyze a new document. This ensures the LLM uses your specific definitions when performing analysis.

3 Small Language Compliance Model: LLM + Theorem Prover = Best of Both Worlds

4 3.1 Rule Representation:

Convert rules/conditions into clear, structured statements. For example break complex rules into atomic components (e.g., "Document must include X" or "Section Y must not contain Z"). Classify rules by type: mandatory inclusions, prohibited content, formatting requirements, etc.

3.2 Consider a hybrid approach:
Formalize clear-cut rules: Use Lean for rules with unambiguous criteria that can be precisely encoded. **Create "proof sketches":** For rules requiring interpretation, create structured reasoning templates that guide LLM analysis. **Develop a domain-specific language:** Create an intermediate representation between natural language regulations and formal Lean theorems. **Progressive formalization:** Start with high-priority rules, formalize those completely, and expand coverage over time. This approach would give you some benefits of formal verification while recognizing the practical limitations of fully formalizing complex regulations. The theorem proving approach is powerful but represents a significant commitment. Parse the regulatory section into a structured inter-

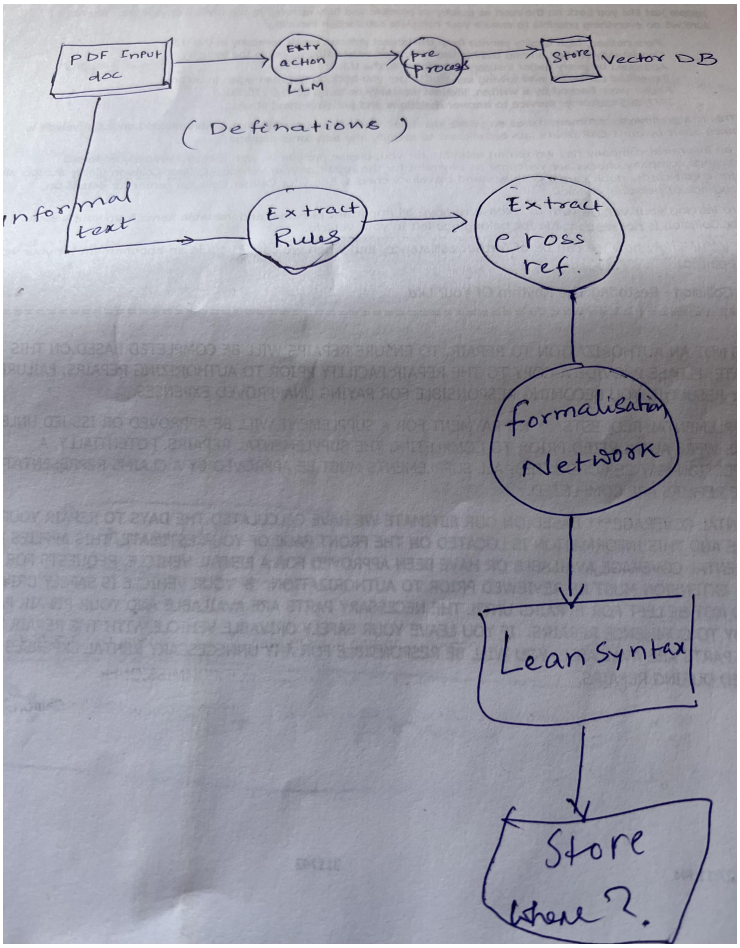

Figure 1: Extraction pipeline.

mediate representation, such as:

```
{
  "rule_id": "4",
  "rule_title": "Conditions for issue of depository receipts",
  "conditions": [
    {
      "id": "4(1)",
      "text": "The Board of Directors of the company intending to issue depository receipts
      "formal_logic": "company.board_passed_resolution_to_issue_dr"
    },
    {
      "id": "4(2)",
      "text": "The company shall take prior approval of its shareholders by a special resolu
      "formal_logic": "company.has_special_resolution_for_dr"
    },
    {
      "id": "4(3)",
      "text": "The depository receipts shall be issued by an overseas depository bank appoi
      "formal_logic": "company.has_overseas_depository_bank",
      "company.has_domestic_custodian_bank"
    },
    {
      "id": "4(4)",
      "text": "The company shall ensure that all applicable provisions of the Scheme and the
      "formal_logic": "company.complies_with_all_rbi_regulations"
    },
    {
      "id": "4(5)",
      "text": "The company shall appoint a compliance overseer and submit the report to the
      "formal_logic": "company.has_compliance_report",
      "committee.has_independent_director"
    }
  ]
}
```


Encode formalized logic in Lean

We write predicates and rules. Assume we're modeling a Company as a structure and each compliance condition as a predicate over it.

Here is some code:

```
structure Company :=
  (board_passed_resolution_to_issue_dr : Prop)
  (has_special_resolution_for_dr : Prop)
  (has_overseas_depository_bank : Prop)
  (has_domestic_custodian_bank : Prop)
  (complies_with_all_rbi_regulations : Prop)
  (has_compliance_report : Prop)
  (committee_has_independent_director : Prop)
```

```
-- Theorem: A company can issue depository receipts if all conditions are satisfied
def eligible_to_issue_depository_receipts (c : Company) : Prop :=
  c.board_passed_resolution_to_issue_dr and
  c.has_special_resolution_for_dr and
  c.has_overseas_depository_bank and
  c.has_domestic_custodian_bank and
  c.complies_with_all_rbi_regulations and
  c.has_compliance_report and
  c.committee_has_independent_director
```

Why LLM + Lean Is Better Suited Here

Deterministic Logic Matters: In law, we want rigorous "provable" compliance—not just "likely based on examples."

Low Tolerance for Error:

Legal violations can lead to penalties. Hypotheses-to-Theories (HtT) is useful for inference, but less reliable for legally binding reasoning.

Auditability And Explainability:

Lean lets you show a proof tree that is checkable by external verifiers.

Modularity and Composability:

Legal clauses can be modularized in Lean (as predicates, theorems, axioms) and reused robustly.

Ideal Hybrid: HtT + Lean + LLM

Consider a 3-phase pipeline:

1. LLM for Rule Extraction

Use an LLM (with HtT or similar method) to extract conditional clauses from natural language statutes and draft candidate Lean predicates.

2. Manual Review And Formalization

A domain expert or compliance engineer reviews/edits the Lean logic.

3. Automated Theorem Proving

Lean performs compliance checking (e.g., "Does Company X satisfy all obligations in Section 4(1)-(5)?"). **Core Components of Auto-Formalization System**

Rule Extraction (LLM or HtT)

Use HtT-style prompting to mine conditionals and logical implications:

"If the Board has not passed a resolution, DRs cannot be issued."

IF NOT Board_Resolution **THEN NOT** Eligible_Issue_DR.

Lean

Lean is a programming language, theorem prover, and interactive proof assistant. It has a vibrant open source community of mathematicians. A huge quest the community is on it to build the math library of the future.

Converting rules into a formal theorem proving approach using Lean comes with both advantages and challenges:

Advantages of the Lean Theorem Proving Approach

Formal verification: Lean provides mathematical certainty about compliance

rather than probabilistic assessment.

Logical rigor: The type theory foundation of Lean forces precise definition of terms and relationships, eliminating ambiguity in rule interpretation.

Composability: Theorems can build upon each other, allowing complex rules to be constructed from simpler proven components.

Reusability: Once formalized, proofs can be reapplied consistently across many documents.

Auditability: The proof chain creates a verifiable record of compliance reasoning.

Challenges to Consider

Translation complexity: Converting natural language regulations into formal logic is extremely difficult, especially for rules with subjective elements or requiring contextual understanding.

Implementation overhead: Developing a comprehensive Lean theorem library for rules represents a significant initial investment.

Maintenance burden: As rules change, maintaining and updating the formal proofs will require specialized expertise.

Expressiveness limitations: Some regulatory concepts may not map cleanly to formal logic (e.g., "reasonable," "appropriate," or context-dependent requirements).

Integration complexity: Connecting document content to formal theorems requires additional abstraction layers.

The theorem proving approach is powerful but represents a significant commitment. It would be ideal for mission-critical compliance where errors carry severe consequences and where rules can be clearly formalized.

5 How to apply to new Document

To apply formalized rules model to a new document under analysis, would follow these steps:

- Document Preprocessing Convert the document to a structured format Identify relevant sections related to depository receipts Extract key statements and declarations
- Entity and Fact Extraction Identify the company and its attributes Extract statements about board resolutions, shareholder approvals Identify mentioned financial institutions (overseas depository banks, custodian banks) Extract information about appointed professionals and committees
- Rule Application Process For each formalized rule in your Lean model: Map extracted facts to the variables in your formal model Check if the document contains sufficient information to satisfy the predicates Apply the theorem to determine compliance
- Practical Implementation Create a mapping layer between document text and formal model concepts For example, statements like "The Board

passed a resolution on June 5, 2025..." map to HasBoardResolution company Check if all required conditions for each theorem are satisfied

- Compliance Determination For formal rules: Apply direct logical verification For semi-formal rules: Use structured templates with LLM assistance to evaluate compliance Record evidence location within the document
- Gap Analysis Identify missing information needed to verify theorems Flag areas where the document fails to address required conditions Generate specific feedback tied to rule violations
- Reporting Generate a compliance report showing: Which rules pass/fail Evidence supporting each determination Gaps and remediation needs

Example Application For a new document that states: The Company's Board of Directors authorized the issuance of depository receipts through Resolution B2023-05 on March 12, 2025. At the Annual General Meeting held on April 15, 2025, shareholders approved this issuance through Special Resolution SR2025-02.

System should:

- Extract facts: Board Resolution exists (B2023-05), Shareholder Special Resolution exists (SR2025-02)
- Map to formal model: HasBoardResolution(company) = true, HasShareholderSpecialResolution(company) = true
- Apply theorem: `board_resolution_required theorem` is satisfied

Record result: `PASS` for rule `DR_PROCESS_01`, with evidence citation For areas where information is missing, your system would flag these as requiring further investigation or documentation. This approach leverages your formal model while providing practical application to real-world documents, giving you mathematically grounded compliance verification with clear explainability.

References:

https://arxiv.org/pdf/2310.07064
https://github.com/google-deepmind/llms_can_learn_rules