Progress Report on the Lean Mission

R. Philip Stetson IV

5 Feb 2025

Abstract

I present two preliminary results:

1. A (prototype) new training data extraction technique.
2. A (partial) understanding of the underlying Mathematics.

This work focuses on problem-solving, querying, and mathematical ability using the Lean programming language, possibly mediated by the LeanDojo framework. The extracted data provides a compiler-invariant representation of fully elaborated Lean programs. For more details, see `README.md` and the references therein.

Contents

1 The Heilmeier Catechism

Responding to a proper subset of the Heilmeier Catechism:

1.1 What are you trying to do? Articulate your objectives using no jargon.

I am trying to solve the problem:

Problem. Create stronger machine-learning-based proof assistant technology, and develop the underlying mathematical theory.

This involves addressing the following subproblems:

- *Problem 0:* What is an optimal framework for understanding learning machines? What should the underlying mathematical theory look like?
- *Problem 1:* Create stronger training data extraction techniques and develop the relevant mathematical theory.
- *Problem 2:* Create stronger learning algorithms and develop the relevant mathematical theory.

Problems 1 and 2 must be solved in tandem. My efforts so far have been focused on *Problem 1*, with Lean serving as an appropriate foundation for exploration. I have preliminary results.

1.2 How is it done today, if at all, and what are the limits of current practice?

The LeanDojo framework [7] provides the most comprehensive training data extraction technique available today. It exports what it refers to as "traces" of tactics, `.lean` files, and constructive theorems (which LeanDojo calls "premises").

These exports are stored as JSON files, each containing localized metadata about its corresponding component.

The form of training data extracted using the LeanDojo framework is obvious from the code of the module `ExtractData.lean`, located in this GitHub repository

```
/--
The trace of a tactic.
-/
structure TacticTrace where
stateBefore: String
stateAfter: String
pos: String.Pos      -- Start position of the tactic.
endPos: String.Pos   -- End position of the tactic.
deriving ToJson


/--
The trace of a premise.
-/
structure PremiseTrace where
fullName: String             -- Fully-qualified name of the premise.
defPos: Option Position      -- Where the premise is defined.
defEndPos: Option Position
modName: String              -- In which module the premise is defined.
defPath: String              -- The path of the file where the premise is defined.
pos: Option Position         -- Where the premise is used.
endPos: Option Position
deriving ToJson


/--
The trace of a Lean file.
-/
structure Trace where
commandASTs : Array Syntax    -- The ASTs of the commands in the file.
tactics: Array TacticTrace    -- All tactics in the file.
premises: Array PremiseTrace  -- All premises in the file.
deriving ToJson
```

While this extraction method is useful, it does not capture fully elaborated Lean programs, nor does it extract the dependency structure that governs their mathematical composition.

1.3 What is new in your approach, and why do you think it will be successful?

Key Idea: The output of my prototype data extraction technique—fully elaborated Lean programs—represents, by definition, all of the mathematically relevant information encoded in Lean. Any alternative training data derived from Lean is a post-processed version of these fully elaborated proof terms.

I propose that we start from this most fundamental representation of Lean programs i.e. mathematical proofs, and from there refine our approach by considering the following:

1. *Modality:* What does a constructive theorem mean, particularly as its context or goal shifts? This should be an explicit consideration in model training.

2. *Constructivism:* A core tenet of constructive mathematics is that the proof method matters, not just the theorem statement itself. Since Lean embodies constructive mathematics, a learning algorithm designed with this in mind may prove advantageous.

3. *Relational Geometry:* The space of all Lean programs is a structured mathematical object. Studying its properties might yield insights into optimizing learning algorithms.

While I cannot predict the success of this approach with certainty, no current training algorithm or data extraction method fully accounts for Lean's foundational mathematical framework—namely, its type theory. It is therefore worth investigating these questions with careful mathematical and computational analysis.

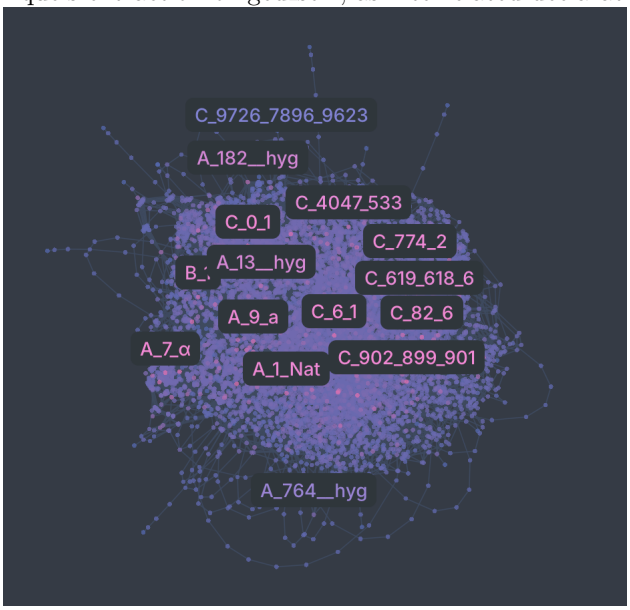
For additional context on Lean's type theory, see Moura's original paper [2] and the recent paper about a Lean 4 compiler programmed in Lean 4 by Mario Carneiro [1].

My favorite reference on the Curry-Howard Isomorphism is the *Study in Logic Lectures* [6]. For more on the theory of proof assistants, interactive theorem proving and formal verification see Henk Barendregt's chapter 18 from the *Handbook of Automated Reasoning* [5].

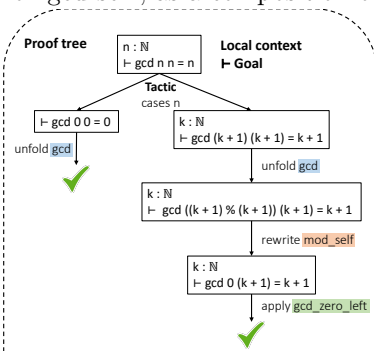
1.3.1 Illustrations

The Lean program gcd_self is a constructive proof of the fact that the greatest common divisor of any natural number and itself is itself.

My proto-technique's extraction of gcd_self, as interrelated declaration-commands:



The LeanDojo extraction of gcd_self, as a composition of tactics (Figure 1 of [7]):



As noted by established researchers in [4], "another way of providing exploration and additional training data [] is required" (page 14). Existing data extraction techniques, including LeanDojo, primarily focus on compositional structures, such as sequences of tactics. However, my approach captures fully elaborated Lean programs, preserving their dependency structure in a way that is mathematically complete. This allows for a richer, more interpretable representation of proofs, which may prove useful for both machine learning applications and foundational studies in theorem proving. While the ultimate impact remains to be determined, this method provides a novel avenue for extracting and structuring training data, potentially offering insights that complement existing frameworks.

1.3.2 Updated Stance on FrontierMath

The primary benchmark for evaluating (what I roughly call text-to-text) machine learning models (particularly LLM's) in mathematical reasoning is the *FrontierMath* dataset. Moving forward, my focus will shift towards:

- Problem-solving, querying, and mathematical ability using the Lean programming language (and *not natural language, as in FrontierMath.*)
- Potential integration with the LeanDojo framework to mediate structured learning.

Nonetheless, the FrontierMath paper remains an important reference [3].

1.3.3 Comparison with LeanDojo and Updated Stance on FrontierMath

A key distinction between my extraction technique and existing methods, such as LeanDojo, lies in the level of abstraction at which training data is captured. LeanDojo extracts tactic sequences, providing insight into proof search behavior. In contrast, my approach exports fully elaborated Lean programs, preserving their dependency structures as directed acyclic graphs (DAGs). This results in a representation that more directly encodes the underlying mathematical relationships between declarations.

Table below summarizes the differences:

Feature	**LeanDojo Extraction**	**This Framework (Lean4Export)**
Extracts Tactic Sequences	Yes	No
Extracts Fully Elaborated Declarations	No	Yes
Captures Proof State Changes	Yes	No
Provides Dependency Structure (DAG)	No	Yes
Works on Arbitrary Input	Yes	Yes

Table 1: Comparison of LeanDojo and Lean4Export

While LeanDojo's focus on tactic sequences aligns well with reinforcement learning applications in theorem proving, my method aims to provide a structurally richer dataset that could support different machine learning paradigms, such as unsupervised learning over mathematical objects.

As acknowledged in the LeanDojo documentation, its current limitations include:

- No extraction from the Lean 4 repository or interaction with its theorems.
- No support for Lean projects using foreign function interfaces (FFI), such as LeanCopilot.
- No ability to process mixed tactic/term-based proofs.
- Extracts only "syntactic theorems" (i.e., Lean constants defined using theorem or lemma), meaning that non-Prop-typed Lean constants are not included.
- Theorems defined using def are not extracted.

1.3.4 Stance on FrontierMath.

Previously, the primary benchmark for evaluating machine learning models in mathematical reasoning was the *FrontierMath* dataset. Moving forward, my focus will shift towards:

- Problem-solving, querying, and mathematical ability using the Lean programming language.
- Potential integration with the LeanDojo framework to mediate structured learning.

Nonetheless, the *FrontierMath* paper remains an important reference [`?frontiermath`], and insights from its benchmarks may still be useful for evaluating progress in automated theorem proving.

1.4 Who cares? If you are successful, what difference will it make?

Developing and understanding stronger machine-learning-based proof assistants may serve as a stepping stone to broader advancements in AI-driven reasoning.

A key observation: The Lean compiler is a rigorous validator of mathematical correctness. Consequently, training models to align with its expectations could generalize beyond theorem proving to other AI applications requiring structured reasoning.

> *Techniques designed to ensure that machine learning models "learn what the compiler wants" could form a generalizable approach to training robust AI systems.*

2 Mathematics

2.1 On the Fundamental Declarations of Lean Programs

A fundamental aspect of Lean programs is their structured hierarchy of declarations. These declarations encode the core logical and computational constructs that define Lean's formal environment.

Understanding the types of declarations that exist is essential for extracting structured data and designing learning algorithms that operate over Lean's mathematical objects. Each declaration falls into one of six primary categories, labeled A through G. These categories serve as an organizational framework for identifying the role of each declaration within the Lean system.

Below, we enumerate the 19 distinct forms of declarations found in Lean programs. These groupings allow for systematic assignment of identifiers, aiding in data extraction and downstream applications such as automated reasoning and machine learning.

Group A - Hierarchical name:

1. $$n' \sqcup \#\mathrm{NS} \sqcup n_0 \sqcup s$$
   (a) n' is an unsigned integer-the unique identifier for the hierarchical name defined by this declaration.
   (b) the string #NS is a mark of the kind of declaration that this is.
   (c) n_0 is the unique integer identifier of a declaration in Group A.
   (d) s is a string.

2. $$n' \sqcup \#\mathrm{NI} \sqcup n_0 \sqcup z$$
   (a) n' is an unsigned integer-the unique identifier for the hierarchical name defined by this declaration.
   (b) the string #NI is a mark of the kind of declaration that this is.
   (c) n_0 is the unique integer identifier of a declaration in Group A.
   (d) z is an integer.

Group B - Universe:

3. $$n' \sqcup \#\mathrm{US} \sqcup n_0$$
   (a) n' is an unsigned integer-the unique identifier for the universe defined by this declaration.
   (b) the string #US is a mark of the kind of declaration that this is.
   (c) n_0 is the unique integer identifier of a declaration in Group B.

4. $$n' \sqcup \#\mathrm{UM} \sqcup n_0 \sqcup n_1$$

(a) n' is an unsigned integer-the unique identifier for the universe defined by this declaration.

(b) the string #UM is a mark of the kind of declaration that this is.

(c) both n_0 and n_1 are unique integer identifiers of declarations in Group B.

5.
$$n' \sqcup \#\text{UIM} \sqcup n_0 \sqcup n_1$$

(a) n' is an unsigned integer-the unique identifier for the universe defined by this declaration.

(b) the string #UIM is a mark of the kind of declaration that this is.

(c) both n_0 and n_1 are unique integer identifiers of declarations in Group B.

6.
$$n' \sqcup \#\text{UP} \sqcup n_0$$

(a) n' is an unsigned integer-the unique identifier for the universe defined by this declaration.

(b) the string #UP is a mark of the kind of declaration that this is.

(c) n_0 is the unique integer identifier of a declaration in Group A.

Group C - Expression:

7.
$$n' \sqcup \#\text{EV} \sqcup z$$

(a) n' is an unsigned integer-the unique identifier for the expression defined by this declaration.

(b) the string #EV is a mark of the kind of declaration that this is.

(c) z is an integer.

8.
$$n' \sqcup \#\text{ES} \sqcup n_0$$

(a) n' is an unsigned integer-the unique identifier for the expression defined by this declaration.

(b) the string #ES is a mark of the kind of declaration that this is.

(c) n_0 is the unique integer identifier of a declaration in Group B.

9.
$$n' \sqcup \#\text{EC} \sqcup n_0 \sqcup \overline{n_0}$$

(a) n' is an unsigned integer-the unique identifier for the expression defined by this declaration.

(b) the string #EC is a mark of the kind of declaration that this is.

(c) n_0 is the unique integer identifier of a declaration in Group A.

(d) $\overline{n_0}$ is a possibly empty finite sequence of unique integer identifiers of declarations in Group B.

10.
$$n' \sqcup \#\text{EA} \sqcup n_0 \sqcup n_1$$

(a) n' is an unsigned integer-the unique identifier for the expression defined by this declaration.

(b) the string #EA is a mark of the kind of declaration that this is.

(c) both n_0 and n_1 are unique integer identifiers of declarations in Group C.

11.
$$n' \sqcup \#\text{EL} \sqcup s \sqcup n_0 \sqcup n_1 \sqcup n_2$$

(a) n' is an unsigned integer-the unique identifier for the expression defined by this declaration.

(b) the string #EL is a mark of the kind of declaration that this is.

(c) s is a string.

(d) n_0 is the unique integer identifier of a declaration in Group A.

(e) both n_1 and n_2 are unique integer identifiers of declarations in Group C.

12.
$$n' \sqcup \#\text{EP} \sqcup s \sqcup n_0 \sqcup n_1 \sqcup n_2$$

(a) n' is an unsigned integer-the unique identifier for the expression defined by this declaration.

(b) the string #EP is a mark of the kind of declaration that this is.

(c) s is a string.

(d) n_0 is the unique integer identifier of a declaration in Group A.

(e) both n_1 and n_2 are unique integer identifiers of declarations in Group C.

Group D - Definition and axiom:

13.
$$\#\text{DEF} \sqcup n' \sqcup n_0 \sqcup n_1 \sqcup \overline{n_0}$$

(a) the string #DEF is a mark of the kind of declaration that this is.

(b) n' is an unsigned integer-the unique identifier for the definition defined by this declaration.

(c) n_0 and n_1 are both integer identifiers declarations in Group C.

(d) $\overline{n_0}$ is a possibly empty finite sequence of unique integer identifiers for declarations in Group A.

14.
$$\#\text{AX} \sqcup n' \sqcup n_0 \sqcup \overline{n_0}$$

(a) the string #AX is a mark of the kind of declaration that this is.

(b) n' is an unsigned integer-the unique identifier for the axiom defined by this declaration.

(c) n_0 is a unique integer identifier for a declaration in Group C.

(d) $\overline{n_0}$ is a possibly empty finite sequence of unique integer identifiers for declarations in Group A.

Group E - Inductive definition:

15.

$$\#\text{IND} \sqcup n_0 \sqcup n' \sqcup n_1 \sqcup n_2 \sqcup \overline{n_0} \sqcup \overline{n_1}$$

(a) string #IND is a mark of the kind of declaration that this is.

(b) n_0 the number of parameters-an unsigned integer.

(c) n' is an unsigned integer-the unique identifier for the inductive definition defined by this declaration.

(d) n_1 is the unique integer identifier for a declaration in Group C.

(e) n_2 is the number of introduction rules-an unsigned integer.

(f) $\overline{n_0}$ is a possibly empty sequence of pairs of unique integer identifiers. A pair of integer identifiers is the concatenation of those two integer identifiers; each of the pairs in the sequence $\overline{n_0}$, if there are any at all, will be of the form $m_0 \sqcup m_1$ where m_0 is the unique integer identifier of a declaration in Group A and m_1 is the unique integer identifier of a declaration in Group C.

(g) $\overline{n_1}$ is a possibly empty finite sequence of unique integer identifiers for declarations in in Group A.

Group G - Format extension:

16.

$$n' \sqcup \#\text{EJ} \sqcup n_0 \sqcup z \sqcup n_1$$

(a) n' is an unsigned integer-the unique identifier for the format extension defined by this declaration.

(b) the string #EJ is a mark of the kind of declaration that this is.

(c) n_0 is the unique integer identifier of a declaration in Group A

(d) z is an integer.

(e) n_1 is the unique integer identifier of a declaration in Group C.

17.

$$n' \sqcup \#\text{ELN} \sqcup z$$

(a) n' is an unsigned integer-the unique identifier for the format extension defined by this declaration.

(b) the string #ELN is a mark of the kind of declaration that this is.

(c) z is an integer.

18.

$$n' \sqcup \#\text{ELS} \sqcup \overline{h}$$

(a) n' is an unsigned integer-the unique identifier for the format extension defined by this declaration.

(b) the string #ELS is a mark of the kind of declaration that this is.

(c) $\overline{h}$ is a possibly empty sequence of UTF8-bytes in hexadecimal.

19.

$$n' \sqcup \#\text{EZ} \sqcup n_0 \sqcup n_1 \sqcup n_2 \sqcup n_3$$

(a) n' is an unsigned integer-the unique identifier for the format extension defined by this declaration.

(b) the string #EZ is a mark of the kind of declaration that this is.

(c) n_0 is the unique integer identifier of a declaration in Group A.

(d) n_1, n_2, n_3 are each a unique integer identifier of a declaration in Group C.

References

[1] Mario Carneiro, *Lean4lean: Towards a verified typechecker for lean, in lean*, 2024.

[2] Leonardo Mendonça de Moura, Soonho Kong, Jeremy Avigad, Floris van Doorn, and Jakob von Raumer, *The lean theorem prover (system description)*, Cade, 2015.

[3] Elliot Glazer, Ege Erdil, Tamay Besiroglu, Diego Chicharro, Evan Chen, Alex Gunning, Caroline Falkman Olsson, Jean-Stanislas Denain, Anson Ho, Emily de Oliveira Santos, Olli Järviniemi, Matthew Barnett, Robert Sandler, Matej Vrzala, Jaime Sevilla, Qiuyu Ren, Elizabeth Pratt, Lionel Levine, Grant Barkley, Natalie Stewart, Bogdan Grechuk, Tetiana Grechuk, Shreepranav Varma Enugandla, and Mark Wildon, *Frontiermath: A benchmark for evaluating advanced mathematical reasoning in ai*, 2024.

[4] Guillaume Lample, Marie-Anne Lachaux, Thibaut Lavril, Xavier Martinet, Amaury Hayat, Gabriel Ebner, Aurélien Rodriguez, and Timothée Lacroix, *Hypertree proof search for neural theorem proving*, 2022.

[5] A.J.A. Robinson and A. Voronkov, *Handbook of automated reasoning*, Handbook of Automated Reasoning, North Holland, 2001.

[6] M.H. Sørensen and P. Urzyczyn, *Lectures on the curry-howard isomorphism*, Studies in Logic and the Foundations of Mathematics, Elsevier Science, 2006.

[7] Kaiyu Yang, Aidan Swope, Alex Gu, Rahul Chalamala, Peiyang Song, Shixing Yu, Saad Godil, Ryan Prenger, and Anima Anandkumar, *LeanDojo: Theorem proving with retrieval-augmented language models*, Neural information processing systems (neurips), 2023.