LEAN 4 in the AI Software Domain

Langze Ye, Xuan Zhang, Jiaheng Xiong
Politecnico di Milano, Italy

***Abstract*—The intersection of artificial intelligence (AI) and programming languages has yielded significant advancements in software development. As AI continues its evolution, the demand for programming languages capable of robust reasoning, formal verification, and efficient modeling grows. LEAN 4, an advanced formal theorem proving language, emerges as a language of profound interest in this context. This paper extensively explores the applications of LEAN 4 within AI software, focusing on its potential to enhance logic-based reasoning, machine learning, and natural language processing. By synergizing formal verification and AI, LEAN 4 paves the way for more dependable, transparent, and secure AI systems.**

***Index Terms*—LEAN 4, artificial intelligence (AI), formal theorem proving language**

I. Introduction

LEAN 4 [1], [2] stands as an advanced formal theorem proving language, equipping developers with potent instruments for formal reasoning and modeling. Rooted in Dependent Type Theory, an influential type system, LEAN 4 empowers developers to articulate intricate logical connections within their code. It deviates from conventional programming languages by not merely focusing on code structure and data types but also on its logical essence.

An outstanding characteristic of LEAN 4 is its support for higher-order logic. This capability permits developers to express intricate logical relationships encompassing first-order logic, predicate logic, and more advanced logical constructs. This versatility bestows LEAN 4 with a distinctive advantage when tackling complex mathematical theorems and logic-based challenges.

Formal proofs constitute another fundamental facet of LEAN 4. This facet enables developers to formally articulate and verify mathematical theorems, algorithmic properties, and software behaviors. Leveraging formal proofs, developers can meticulously establish the correctness of their code, substantially mitigating the likelihood of errors and defects.

LEAN 4 takes strides in facilitating interactive theorem proving. This approach empowers developers to construct proofs incrementally, engaging with the system in a step-by-step fashion to craft intricate arguments progressively. This interactive paradigm not only enriches developers' comprehension of the problem but also assures the precision of the ensuing proof.

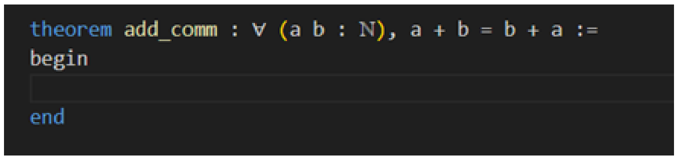
In the realm of AI software development, these innate attributes of LEAN 4 furnish developers with potent instruments for formal reasoning and modeling. LEAN 4 can be effectively harnessed to articulate the intricacies of complex algorithms and models, verify the logical underpinnings of the reasoning process, and guarantee the correctness and security of AI systems. The prowess of LEAN 4 in formal reasoning and proof construction renders it an indispensable asset in the arsenal of AI software developers.

In this study, we conducted an assessment of the Lean 4 language as an interactive theorem prover, aiming to evaluate its advantages in the realm of mathematical theorem proving.In our investigation, we explored the potential applications of Lean 4 in the domain of AI software development. These applications encompass areas such as logical reasoning, machine learning, and natural language processing, among others.

II. LEAN 4 in Logical Reasoning - Mathematical Theorem Proving

A. the law of exchange for the addition of natural numbers

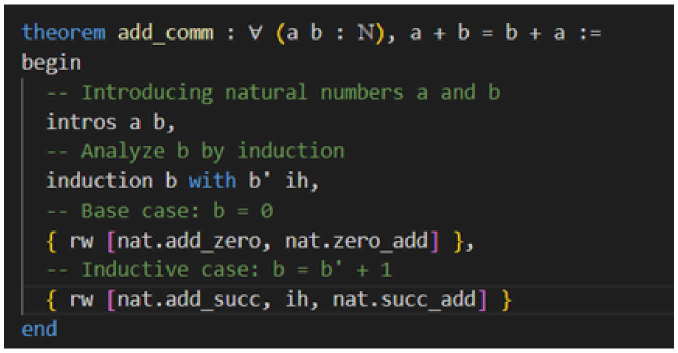
In order to demonstrate the formal theorem proving power of LEAN 4, we can consider a simple mathematical theorem, such as the law of exchange for the addition of natural numbers as Fig. 1. In LEAN 4, we first need to represent the problem formally. We can define a function that represents the addition of natural numbers and state this exchange law theorem.



```
theorem add_comm : ∀ (a b : ℕ), a + b = b + a :=
begin

end
```

Fig. 1. Represent the law of exchange for the addition of natural numbers in lean4

Then we need to fill in the proof steps between 'begin' and 'end'. In Lean 4, proofs are constructed by step-by-step logical reasoning and application of known theorems. Each step of the proof requires sound reasoning based on logical rules. For the exchange law theorem, the codes are as Fig. 2



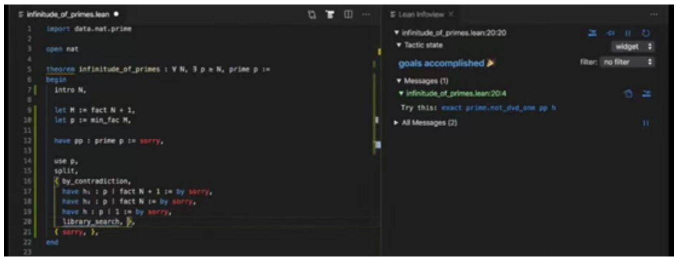
```
theorem add_comm : ∀ (a b : ℕ), a + b = b + a :=
begin
  -- Introducing natural numbers a and b
  intros a b,
  -- Analyze b by induction
  induction b with b' ih,
  -- Base case: b = 0
  { rw [nat.add_zero, nat.zero_add] },
  -- Inductive case: b = b' + 1
  { rw [nat.add_succ, ih, nat.succ_add] }
end
```

Fig. 2. Proof steps for the exchange law theorem

In this case, at first, natural numbers a and b are introduced and they are the most fundamental assumptions needed for the proof process. Then we use induction to analyze the variable b. INTRODUCTION is used to make a mathematical induction that will analyze various scenarios of the variable b and will apply the appropriatereasoning steps in each case. $rw[nat.add_zero, nat.zero_add]$ means when b is 0, we use the rw (rewrite) strategy to apply a known mathematical equation by replacing the terms on both sides of the equation. Here, we use the properties of addition of natural numbers, where $nat.add_zero$ means that the result of adding 0 to a natural number, and $nat.zero_add$ means that the result of adding 0 to any natural number is still that natural number. Finally, when b is b' + 1. We use the rewrite strategy again, applying the properties of addition of natural numbers and the induction hypothesis (ih), respectively. here, $nat.add_succ$ denotes the definition of the successor of addition of natural numbers, and $nat.succ_add$ denotes the result of adding the successor number to the other. Then, we're done with proof. With this example, we can find that each step must be logically rigorous and requires a proof based on a theorem already in the library. In fact, many complex mathematical theorems can be broken down into smaller steps using the lean4 Theorem Prover.

B. *Infinity of Primes*

We can also try to prove the infinity of primes using lean4. We need to prove that for every natural number n, there is a prime number greater than n. Our proof procedure can be based on Euclid's theorem. We can let p be any prime factor of n! + 1. If p is less than n, it divides n!. Since it also divides n! + 1, it divides 1, a contradiction. Hence p is greater than n. In LEAN 4, we need to formalize the problem first, after which we write the proof process and logic for each step. [3]



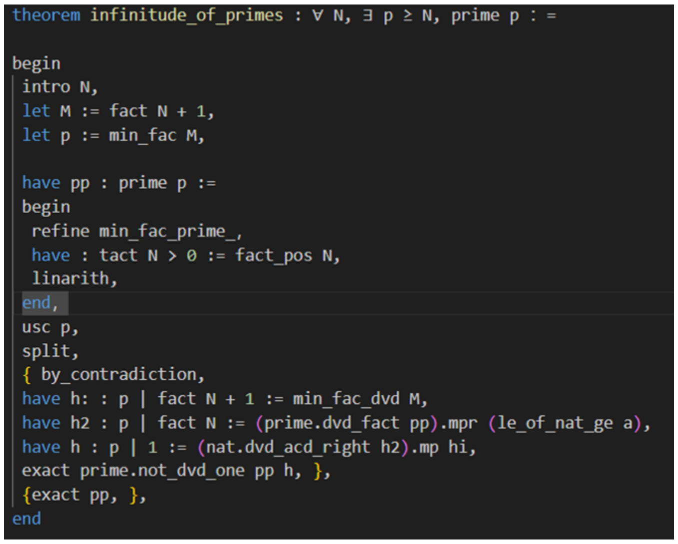
```
infinitude_of_primes.lean

import data.nat.prime

open nat

theorem infinitude_of_primes : ∀ N, ∃ p ≥ N, prime p :=
begin
  intro N,

  let M := fact N + 1,
  let p := min_fac M,

  have pp : prime p := sorry,

  use p,
  split,
  { by_contradiction,
    have h₁ : p ∣ fact N + 1 := by sorry,
    have h₂ : p ∣ fact N := by sorry,
    have h : p ∣ 1 := by sorry,
    library_search, },
  { sorry, },
end
```

```
Lean Infoview
▼ infinitude_of_primes.lean:20:20
 ▼ Tactic state
 goals accomplished
 ▼ Messages (1)
 ▼ infinitude_of_primes.lean:20:4
  Try this: exact prime.not_dvd_one pp h
▶ All Messages (2)
```

Fig. 3. Proof steps for the infinity of primes

What's more, the interactivity of the lean 4 theorem prover often helps us a lot. For example, in Fig. 3, we have written a general proof framework but missing theorem support. Every red "sorry" need to be replaced by a known theorem in the library or other effective methods of proof. We can try to type "library_search" or "suggest" to find some useful theorems in the library for our proving. On the right of the picture, LEAN 4 calls a theorem inside the library in LEAN infoview to help us refine our proof logic. These constant interactions with the AI often provoke us to think more about the problem, which opens up our minds and contributes to problem solving.

In Fig. 4,we formalized the question, define the parameters(p,N,M) we need, write each step of the proof and write theorem supports for each step after":=". So that we have completed all the proof steps.



```
theorem infinitude_of_primes : ∀ N, ∃ p ≥ N, prime p : =

begin
 intro N,
 let M := fact N + 1,
 let p := min_fac M,

 have pp : prime p :=
 begin
  refine min_fac_prime_,
  have : tact N > 0 := fact_pos N,
  linarith,
 end,
 usc p,
 split,
 { by_contradiction,
 have h: : p | fact N + 1 := min_fac_dvd M,
 have h2 : p | fact N := (prime.dvd_fact pp).mpr (le_of_nat_ge a),
 have h : p | 1 := (nat.dvd_acd_right h2).mp hi,
 exact prime.not_dvd_one pp h, },
 {exact pp, },
end
```

Fig. 4. Complete proof steps for the infinity of primes

It's also worth noting that, when dealing with very complex tasks, we can divide the task into many small red "sorry" and hand over to different people to complete the proof in parallel. Thus LEAN 4 Interactive theorem provers make the proof of mathematical theorems more efficient and rigorous.

C. *Proof of The Schröder-Bernstein Theorem*

The Schröder-Bernstein Theorem: If there exists an injection $\phi : A \to B$ from set A to set B and an injection $\psi : B \to A$ from set B to set A, then sets A and B have the same cardinality, i.e., they are equipotent.

Proof:

Let $\phi : A \to B$ and $\psi : B \to A$ be injections. We will construct a bijection between sets A and B.

Define the sets A_0 and B_0 as follows:

$$A_0 = \{a \in A \mid \psi(\phi(a)) \neq a\}$$
$$B_0 = \{b \in B \mid \phi(\psi(b)) \neq b\}$$

Now, define the sets A_1 and B_1 as:

$$A_1 = A \setminus A_0$$
$$B_1 = B \setminus B_0$$

We will construct a bijection $\alpha : A \to B$ as follows:

$$\alpha(a) = \begin{cases} \phi(a) & \text{if } a \in A_1 \\ \psi^{-1}(a) & \text{if } a \in A_0 \end{cases}$$

We will now prove that α is a bijection.

Injectivity (One-to-One):

Let a_1 and a_2 be two distinct elements in A. We need to show that $\alpha(a_1) \neq \alpha(a_2)$.

- If both a_1 and a_2 are in A_1, then $\alpha(a_1) = \phi(a_1)$ and $\alpha(a_2) = \phi(a_2)$, and since ϕ is injective, we have $\alpha(a_1) \neq \alpha(a_2)$.

- If both a_1 and a_2 are in A_0, then $\alpha(a_1) = \psi^{-1}(a_1)$ and $\alpha(a_2) = \psi^{-1}(a_2)$, and since ψ^{-1} is injective, we again have $\alpha(a_1) \neq \alpha(a_2)$.

- If one element is in A_1 and the other in A_0, without loss of generality, assume $a_1 \in A_1$ and $a_2 \in A_0$. Then, $\alpha(a_1) = \phi(a_1)$ and $\alpha(a_2) = \psi^{-1}(a_2)$, and since $\phi(a_1)$ and $\psi^{-1}(a_2)$ are in different subsets of B, we have $\alpha(a_1) \neq \alpha(a_2)$.

Therefore, α is injective.

Surjectivity (Onto):

Let b be an arbitrary element in B. We need to show that there exists an element a in A such that $\alpha(a) = b$.

- If $b \in B_1$, then $\alpha^{-1}(b) = \phi^{-1}(b)$, and since ϕ is surjective, there exists $a \in A$ such that $\phi(a) = b$, and thus, $\alpha(a) = b$.

- If $b \in B_0$, then $\alpha^{-1}(b) = \psi(\psi^{-1}(b))$, and since ψ is surjective, we have $\alpha^{-1}(b) = b$, so we can choose $a = \psi^{-1}(b)$, and thus, $\alpha(a) = b$.

Therefore, α is surjective.

Since α is both injective and surjective, it is bijective.

Conclusion:

We have constructed a bijection $\alpha : A \to B$, which implies that sets A and B have the same cardinality. Therefore, the Schröder-Bernstein Theorem is proved.

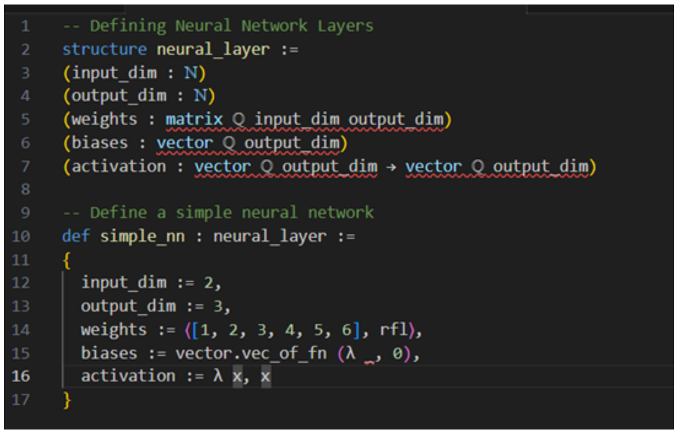
III. D. LEAN 4 on Applications of Machine Learning

In addition to being used for mathematical theorem proving, the formal proof capabilities of LEAN 4 can be used in a wide range of machine learning applications, such as representing neural network structures and formal machine learning algorithms. The potential of LEAN 4 will be unfolded in detail below in conjunction with the code.

A. *To represent of neural network structures*

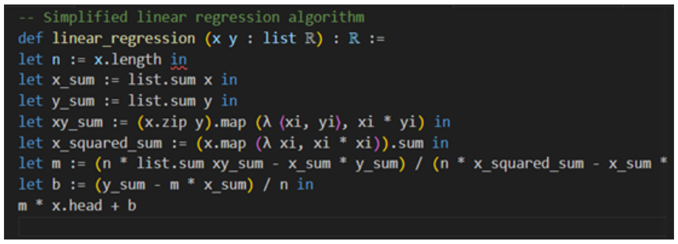
Neural networks are a key component in machine learning.Lean 4 can be used to formally represent the structure of a neural network, ensuring that its specific layers, activation functions, etc. are defined and related correctly. A simplified example is as Fig. 5

In Fig. 5, we use Lean 4's record type **neural_layer** to define the structure of a neural network layer. We define attributes such as input dimensions, output dimensions, weights, biases, and activation functions.The "activation function" is a function that takes a vector of rational numbers with length output_dim (the outputs from the layer) and returns a new vector of rational numbers with length output_dim. This function represents the activation function applied to the outputs of the layer.Then we create a simple neural network layer named simple_nn. It has 2 input neurons, 3 output neurons, specific weight values, biases set to 0, and uses the identity function as the activation function. Finally, We create a simple neural network layer **simple_nn**.



```
-- Defining Neural Network Layers
structure neural_layer :=
(input_dim : N)
(output_dim : N)
(weights : matrix Q input_dim output_dim)
(biases : vector Q output_dim)
(activation : vector Q output_dim → vector Q output_dim)

-- Define a simple neural network
def simple_nn : neural_layer :=
{
  input_dim := 2,
  output_dim := 3,
  weights := ⟨[1, 2, 3, 4, 5, 6], rfl⟩,
  biases := vector.vec_of_fn (λ _, 0),
  activation := λ x, x
}
```

Fig. 5. represent of neural network structures in lean 4



```
-- Simplified linear regression algorithm
def linear_regression (x y : list R) : R :=
let n := x.length in
let x_sum := list.sum x in
let y_sum := list.sum y in
let xy_sum := (x.zip y).map (λ ⟨xi, yi⟩, xi * yi) in
let x_squared_sum := (x.map (λ xi, xi * xi)).sum in
let m := (n * list.sum xy_sum - x_sum * y_sum) / (n * x_squared_sum - x_sum *
let b := (y_sum - m * x_sum) / n in
m * x.head + b
```

Fig. 6. Formalized Machine Learning Algorithms in Lean 4

B. *2. Formalized Machine Learning Algorithms*

Lean 4 can be used to formalize the nature and behavior of machine learning algorithms. For example, we might consider formalizing a simple linear regression algorithm:

Fig. 6 is a simple linear regression algorithm. The first line defines a function named linear_regression that takes two lists of real numbers (x and y) as input and returns a real number. In this context, x represents the independent variable, and y represents the dependent variable. Then we create a local variable n and assigns it the length of the list x. In the context of linear regression, n represents the number of data points in the dataset. After that, we compute the sum of all elements in the lists x and y respectively. x_sum and y_sum represent the sum of the independent and dependent variables, respectively. xy_sum is a fonction which combines the elements of x and y into pairs, then multiplies each pair's elements together. Its result is a list of products of corresponding x and y values. We have also two lines to calculate the parameters of the linear regression model where m represents the slope of the regression line, and b represents the y-intercept. The formulas used are based on the mathematical formulas for linear regression. Finally, the last computes the predicted value of y using the linear regression model. It multiplies the slope m by the first element of x (.head gives the first element of a list) and adds the y-intercept b.

By formalizing it and proving some properties, we can ensure that the algorithm is correct. In addition, we can formalize other machine learning algorithms and select the optimal solutions of various algorithms through an interactive theorem prover, depending on the actual situation.

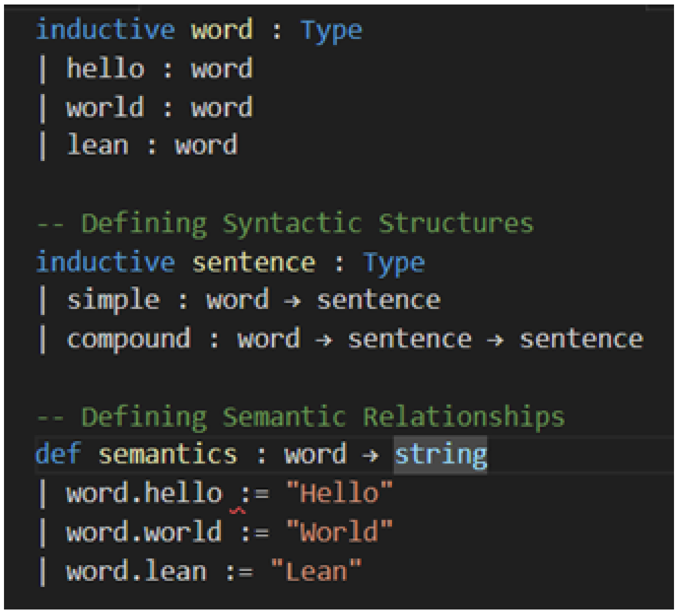
In brief, with these examples, we can see the potential of Lean 4 in machine learning. Lean 4 helps improve the reliability and trustworthiness of machine learning systems.

IV. Lean 4 in Natural Language Processing

Lean 4 is also widely used in the field of Natural Language Processing (NLP) to support formal representations of natural language. It also can help with semantic analysis.

A. *Formal representation of natural language*

In natural language processing, the formal representation of text is key. lean 4 can be used to define vocabulary, syntactic structures and semantic relationships in natural language. Fig. 7 simplified example,



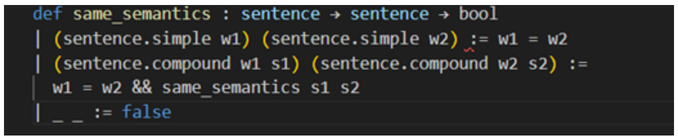
```
inductive word : Type
| hello : word
| world : word
| lean : word

-- Defining Syntactic Structures
inductive sentence : Type
| simple : word → sentence
| compound : word → sentence → sentence

-- Defining Semantic Relationships
def semantics : word → string
| word.hello := "Hello"
| word.world := "World"
| word.lean := "Lean"
```

Fig. 7. Formal representation of natural language in Lean 4

In Fig. 7, we define a simplified vocabulary of ”hello”, ”world” and ”lean”. Then, we define syntactic structures, where sentences can be simple vocabulary or compound sentences. Finally, we define a semantic function that maps the vocabulary to the corresponding string.

B. *2. Semantic analysis*

In addition to performing basic formal natural language, he can also be used to assist in semantic analysis. Fig. 8 is an example for determining whether two sentences have the same semantics.



```
def same_semantics : sentence → sentence → bool
| (sentence.simple w1) (sentence.simple w2) := w1 = w2
| (sentence.compound w1 s1) (sentence.compound w2 s2) :=
  w1 = w2 && same_semantics s1 s2
| _ _ := false
```

Fig. 8. Semantic analysis in Lean 4

In Fig. 8, we define a function **same_semantics**, used to determine whether two sentences have the same semantics. Here is just a simplified example which demonstrates basic syntax. Implementing more complex semantic analysis and reasoning is also feasible, but requires more complex syntactic structures.

V. Conclusion and Analysis

With the above example, we can see that LEAN 4’s impact on the AI domain lies in its formalization capabilities and interactivity. Firstly, LEAN 4 can be used as an interactive theorem prover to prove complex mathematical problems, while the interactivity it provides can deepen one’s thinking about the problem. What’s more, thanks to its formalization capabilities, LEAN 4 can be used to solve complex problems in other domains as well. For example, in the field of machine learning, LEAN 4 can be used to represent neural network structures, formalize machine learning algorithms, verify model properties, etc. Similarly, LEAN 4 can formalize natural language, analyze semantics, etc., which is helpful in the field of natural language processing (NLP).

In conclusion, LEAN 4 has a number of advantages for applications in the AI software space like its formalized capacity, rigor and credibility, and its interactive nature also helps reduce manual labor.

However, every coin has 2 sides, we also realized its limitations in the process of learning LEAN. For example,although Lean has an active community that provides a wealth of learning resources and discussion platforms [4], LEAN 4 has a relatively steep learning curve and takes some time and effort to master. At the same time, the configuration of the compilation environment and the import of libraries are complex. In practical applications, specialized training and resources may be required. What’s more, although LEAN 4 supports formalization, facing complex problems will require more expertise and programming ability. LEAN’s interactive proof system also has some limitations in solving complex problems.

All in all, in my opinion, the potential for LEAN 4 to be used in the AI software space remains huge. As technology evolves, LEAN 4 may further enhance the ability to automate formalisms, thereby reducing manual intervention and better accommodating the needs of large-scale systems. Specific LEAN 4 libraries may also emerge for the AI domain, including formal machine learning algorithms, model validation tools, etc., to simplify applications. If future versions of LEAN can be stabilized with constant updates, it will be one of the most promising tools of the future!

References

[1] “Lean official website,” https://leanprover.github.io/about/.

[2] “Lean 4 quick start,” https://github.com/leanprover/lean4/blob/master/doc/quickstart.md.

[3] “Prove the infinity of primes by lean 4,” https://www.bilibili.com/video/BV1wU4y1475g/?spm_id_from=333.337.search-card.all.click&vd_source=e1e93e424f66b968a618cde73640d75a.

[4] “Lean prover zulipchat room,” https://leanprover.zulipchat.com/.