Learning Lean, A DRP Project

Eloise Yvelise Freydier

Mentor: Jérémie Turcotte

DRP Winter 2024, McGill University Mathematics & Statistics

1. Introduction

This winter semester, as part of the Directed Reading Program (DRP), we delved into theorem proving using Lean.

Lean is a proof assistant designed to formalize mathematical notions and support the verification of various theorems and statements. This programming language facilitates a solid understanding of proofs through step-by-step validation.

Lean relies on type theory, which classifies different kinds of terms to define rules regarding their manipulation. The language provides an interactive proving platform with a set of known statements and theorems, along with a set of goals, to guide the user through each step of the proof.

2. Basics

For this research project, we followed *Mathematics in Lean* starting with the early modules as it is necessary to be familiar with the basics of the Lean syntax.

Below are common keywords used in Lean and their utilities:

- def : keyword to define functions
- intro: keyword to introduce a new hypothesis
- theorem: keyword to define theorems
- by: keyword to indicate the beginning of the proof
- rw[h]: keyword to rewrite the goal using a given statement “h”
- apply: keyword to reduce the goal using a theorem or a hypothesis
- refl: keyword to solve by reflexivity
- unfold: keyword to replace a defined term with its body in the goal or hypothesis
- cases: keyword to break down into different cases to do case analysis

- left/right: keyword to handle disjunctions like logical OR.

3. Demonstration of a simple Lean proof

To fully understand the effectiveness and clarity of Lean for proofs, let's examine an example involving full binary trees (where each parent node or internal node has either two or no children). This proof will focus on the relationship between the number of leaves in a full binary tree and its number of internal nodes.

Theorem: Let T be a nonempty, full binary tree.
Then: if T has I internal nodes, the number of leaves is $L = I + 1$.

3.1. Non-Lean Proof

The following proof by induction comes from *Proof of Full Binary Tree Theorem*.

We will prove the statement using induction on the number of internal nodes I . Let S be the set of all integers $I \geq 0$ such that if T is a full binary tree with I internal nodes, then T has $I + 1$ leaf nodes.

Base Case:
When $I = 0$, it means that the tree consists of one node without any children, as it is a full binary tree. So, it has exactly 1 leaf node. Therefore, $0 \in S$.

Induction Hypothesis:
Assume that for some integer $K \geq 0$, every integer I from 0 to K is in S. This means that for any full binary tree T with I internal nodes where $0 \leq I \leq K$, the tree T has $I + 1$ leaves.

Induction Step:
Consider a full binary tree T with $K + 1$ internal nodes. The root of T has two subtrees, L and R, which have I_L and I_R internal nodes respectively. Since neither L nor R can be empty and all internal nodes in L and R were also internal nodes in T, we have:

$$K + 1 = I_L + I_R + 1$$

Using the induction hypothesis, the subtree L has $I_L + 1$ leaves, and the subtree R has $I_R + 1$ leaves. Therefore, the total number of leaf nodes in T is:

$$I_L + 1 + I_R + 1 = I_L + I_R + 2 = K + 2$$

Thus, T has $K + 2$ leaf nodes, which means $K + 1 \in S$. By the principle of Mathematical Induction, the set S includes all integers $I \geq 0$. Hence, the statement holds for all $I \geq 0$. □

3.2 Lean Proof

A challenging aspect of Lean proofs is that they require a proper setup. To ensure a solid proof, or even just a compiling Lean file, objects must be correctly defined, including types, and any auxiliary theorems or properties used in the main proof must also be rigorously proved.

Note: code will be highlighted while comments will be left default.

Definition of a binary tree (btree for simplicity)
Notice here that a tree is defined as : current node, its left subtree, its right subtree and x → y → is a triple.

```
inductive btree
| empty : btree
| node : btree → btree → btree
```

Function to determine if the binary tree is full which returns a boolean

```
def is_full : btree → Prop
| empty := true
| (node empty empty) := true
| (node l r) := is_full l ∧ is_full r
| _ := false
```

Function to count the number of leaves in a binary tree which returns a natural number

```
def num_leaves : btree → ℕ
| empty := 0
| (node empty empty) := 1
| (node l r) := num_leaves l + num_leaves r
```

Function to count the number of internal nodes in a binary tree

```
def num_internal_nodes : btree → ℕ
| empty := 0
| (node empty empty) := 0
| (node l r) := 1 + num_internal_nodes l + num_internal_nodes r
```

Now, onto the main proof of the theorem:

```
theorem full_tree_leaves_internal (t : btree) (h : is_full t) : num_leaves t =
num_internal_nodes t + 1 :=
begin
Perform induction on the tree structure. Note: ih means inductive hypothesis.
induction t with l r ihl ihr
Case: t is empty (base case).
An empty tree cannot be full, hence we get a contradiction
    unfold is_full at h
    contradiction
Cases where t is a node with subtrees l and r
cases l
Case: l is empty
    cases r
    Case: r is also empty, so t is a single node (base case)
    unfold num_leaves num_internal_nodes
    refl
    Case: r is not empty, but l is empty, so t is not full
    unfold is_full at h
    contradiction
Case: l is not empty
    cases r
    Case: r is empty, but l is not, so t is not full
    unfold is_full at h
    contradiction
Case: both l and r are not empty (inductive step) Since both children are non-empty,
use induction hypothesis
    unfold is_full at h
    Apply induction hypothesis to left and right subtrees
    have hl := ihl
    have hr := ihr
    Calculate the number of leaves and internal nodes
    unfold num_leaves num_internal_nodes
    Substitute induction hypothesis results
    rw [hl, hr]
    Simplify the arithmetic expression
    ring
end
```

4. Non-Lean Proof vs Lean Proof

The above proofs have the same goal but are designed in two very different ways. The non-Lean proof relies on simple explanations, while the other is code-based.

First, let's examine the perks and disadvantages of the non-Lean proof. At first glance, this proof is much shorter than the Lean one, though not necessarily in terms of word count. It explains its reasoning clearly and relies on algebraic manipulations to achieve its purpose. The main disadvantage of this method is that validating the proof requires either a professional or an existing proof for comparison.

On the other hand, the Lean proof is much longer and requires prior knowledge and understanding of how Lean works. Not only is it preferable to have some experience with coding, but Lean syntax is also not beginner-friendly due to its precise keywords. To effectively use the Lean method, it is recommended to have at least a few weeks of experience with the platform. Despite these challenges, Lean is a very effective method for proving statements since it can clearly illustrate manipulations when the purpose of each tactic is understood. Additionally, Lean's built-in correctness checker verifies the validity of the proof, which is very beneficial for self-learning without needing external help for proofreading.

At last, the choice between these methods depends on the user's familiarity with formal proof systems and their specific needs for proof verification and self-learning.

5. Conclusion

In this DRP project, we explored the application of Lean, a proof assistant, to formalize and verify mathematical theorems.

This past winter semester, Lean underscored its utility not just as a proof assistant but also as an educational tool. It demonstrates a disciplined approach to theorem proving, making explicit each logical step and its justification. This methodical process benefits both understanding and confidence in mathematical reasoning.

Finally, our journey with Lean in this DRP project guided by M.Turcotte has been enriching, equipping us with valuable skills in formal verification and proof writing. This project has not only expanded our technical abilities but also deepened our appreciation for the rigor and beauty of formal mathematics.