Day - 2 : Propositions as Types, Proofs as Objects (Curry-Howard Correspondence)

Touseef Haider

May 13, 2025

2.1. Propositions are Types

In mathematics, a *proposition* is a statement that can be either true or false.

- $2 + 2 = 4$ is a true proposition.
- *The Earth is flat* is a false proposition.
- *x is an even number* is a proposition that depends on the value of x.

In Lean (and other systems based on what's called *dependent type theory*, this idea is taken a step further : *Every proposition is considered a type.* This is the core of the *Curry-Howard correspondence (or Curry-Howard-Lambek correspondence).* It draws a direct analogy between logic and computation.

If a proposition is like `P : Prop` (This is how Lean denote that P is a proposition). You can think of `P` not just as a statement, but as a *type*.

- The proposition $5 > 3$ can be thought of as a type.
- The proposition "for all natural numbers n, $n + 0 = n$" can be thought of a type.

This might seems strange. What does it mean for a statement like $5 > 3$ to be type? This leads directly to our next point about proofs.

The key to making " propositions as type" is to see what inhabits these types.

2.2. Proofs are Objects (Terms) of that Type

If a proposition `P` is a type, then what is an *object* (or element or inhabitant) of type `P` ?

A proof of `P` is an object of type `P`. Let's break this down:

- Type `P` : This is our proposition. Think of it as a specification or a challenge. For example, the type `IsEven 6` represents the proposition "6 is an even number".
- Object `p:P` : If we can construct an object `p` that has the type `P`, then `p` is the proof of the proposition `P`.

So,

- If the type `P` (our proposition) has at least one object/term/element, then the proposition `P` is considered true (because a proof exists).
- If the type `P` is empty means it has no object/term/element, then the proposition `P` is considered false (because no proof exists).

Proposition: $2 + 2 = 4$

- Type : $2 + 2 = 4$. This is literally a type in Lean.
- Proof/Object: Lean has a way to construct a term, let's call it proof_of_equality such that proof_of_equality : $2 + 2 =$ The existence of this term means the propitiation is true. Often, for simple definitional equalities, this proof term is something like `rfl` (which we will see later).

Proposition: The Earth is flat.

- Type : The Earth is flat .
- Proof/Object : We would expect this type to be empty. There is no valid construction/proof that can be assigned this type within a consistent mathematical framework that includes facts about the Earth.

This is the essence of the Curry-Howard correspondence:

- Propositions $\leftrightarrow$ Types
- Proofs $\leftrightarrow$ Programs/Terms/Objects
- Provability $\leftrightarrow$ Inhabitation (is there a term of that type?)

It connects logic (propositions, proofs) with computation (types, programs). When we write a proof in Lean, we are essentially constructing a term of a specific type. The tactics we use are tools to help us build this term.

2.3. Analogy: The EvenNumber

Imagine we define a type called IsEven. This type is not just for any number, but it is a type that can only be inhabited by numbers that are actually even.

- Proposition as a Type: Let's say we have a specific number, like 6. The proposition 6 *is an even number* can be represented by a type. We could think of this type as IsEven 6.
- Proof as an object : How would we prove that 6 is even? We would show that $6 = 2 * 3$. This demonstration, this piece of evidence, is the proof. In a system like Lean, we construct a specific *proof object* (let's call it proof_that_6_is_even) which fundamentally relies on the definition of even (e.g. n is even if there exists an integer k such that $n = 2 * k$. This proof_that_6_is_even would then have the type IsEven 6. So proof_that_6_is_even : IsEven 6.
- Construct with an Odd Number: Now consider the number 7. The proposition 7 *is an even number* would correspond to the type IsEven 7. Since 7 is not even, we would not be able to construct a valid proof object for this type. The type IsEven 7 would be uninhabited.

Connecting to Propositions are Types:

- The type IsEven n (where n is some number) is like asking the question or stating the challenge: *Is n an even number*?
- An object that successfully inhabits the type IsEven n is the answer *Yes, and here is why. (here is the evidence/proof.*

This is slightly different from saying 6 : Nat (6 is a type Natural Number). `Nat` is a collection of numbers. IsEven 6 is more like a certification or a proof that specifically confirms the property that 6 is even number.

Why is this useful? When we write theorem my_theorem : P := by ... proof ... Lean is checking that your proof actually constructs a valid term of type `P`. If it does, the theorem is proved.