

# A Tentative Explanation of Types in Lean

Here is my attempt at a simple explanation of types in Lean and especially of the “Propositions as Type” paradigm in Lean. The vocation of this document is not to be rigorous (it definitely isn’t) but to understand the high-level ideas of how types are structured.

---

## Type levels to solve Russel Paradox

A type can be thought of as a set. An `int` in a traditional programming language can roughly be thought of as the set of all integers (or at least all integers that can be written on `X` bits). To avoid confusion on the language as the word `type` might get overloaded, I will say that “`32` belongs to (or is an element of) the set of `ints`” instead of “`32` has type `int`”.

**Disclaimer:** I believe that confounding type and set is not exact but I used it as both a useful approximation and a way to avoid the word ‘type’ that is sometimes overloaded, leading to confusion.

In python, `type(32)` will inform you that `32` is an element of the set of `<int>`. `type(int)` will inform you that the set of `ints` is an element of the set of `<type>`. Finally if you run the `type(type)`, you get that the set of `types` is an element of the set of ... `<type>`. In ZFC (the most common set axiomatic), this is an issue: a set cannot contain itself. (From my understanding, a coherent set theory where a set belongs to itself exists but for the non-rigorous document this is, it is beyond the point so let’s accept that a set containing itself is not desirable).

Lean solves this problem by introducing a stratification of sets, denoted by `Sort x` where `x` is the strata. For instance:

- The strata of `Nat`, the set of all natural numbers, is `1`. `Nat` is a `Sort 1` set.
- `Nat` belongs to the set `Type 0` (Lean usually shortens `Type 0` to just `Type`). `Type 0` has strata `2`, it is a `Sort 2` set.
- This keeps going: `Type 0` is an element of `Type 1`, which is `Sort 3` etc... In general `Type x` is `Sort x+1`.

Each level is called a **Universe**. Note that only sets have a `Sort` level. `3` for instance is an element of `Nat` but it’s not a set so it doesn’t have a `Sort` level. It’s just data.

---

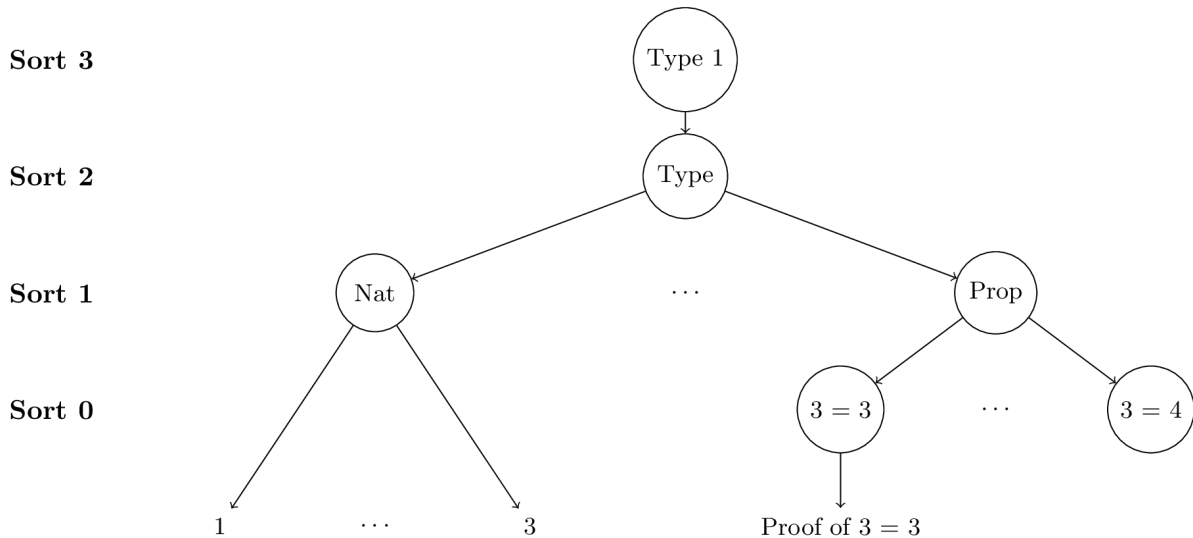
## Proposition as Types

Remember that a type can roughly be thought of as a set. “Proposition as Types” means that propositions are sets. This means that a single proposition (e.g. `3 = 3`) is a set. This is not to be confused with `3 = 3` belongs to the set of `Propositions` (which is also true but less surprising).

- The object  $3=3$  **IS** a set that is in universe **Sort 0**.
- The object  $3=3$  belongs to the set **Prop** of all propositions that is in the universe **Sort 1**.

So what is contained inside a proposition? **Its proof!** For instance, a proof that  $3 = 3$  would belong to the set  $3 = 3$ . In other words, a proposition **IS** the set of its proofs.

The situation so far is summarized in the following figure. Elements in a circle are sets while leafs aren't.



## Proof irrelevance

The concept of Proof irrelevance that Lean implements means that two proofs of a proposition are considered equal. If  $h1$  and  $h2$  are two proofs of  $p$  then  $h1 = h2$  for Lean (even if for a human, they may seem like different proofs). This means that each proposition can either be:

- **A singleton** if it admits a proof, i.e. if it's a **True** statement.
- **The empty set**, i.e. it's a **False** statement.

This allows us to treat implication as a function (hence the ambiguity of what the arrow means in  $P \rightarrow Q$  in lean). Let's consider two propositions  $P$  and  $Q$  and consider the 4 cases:

1. **P true, Q true:** both  $P$  and  $Q$  are singleton hence there is a unique function mapping the proof of  $P$  to the proof of  $Q$ .
2. **P false, Q true:**  $P$  is empty so a function mapping all elements of  $P$  to an element of  $Q$  exists.
3. **P false, Q false:** same as previous case.

4. **P true, Q false:** P contains an element but Q doesn't so there is no way to map the element of P to the element of Q.

Similarly, a bijection between (the sets) P and Q only exists iff  $(P \Leftrightarrow Q)$ .

---

## Type lifting

Types (sets) can be lifted from their universe (Sort x) to a higher universe (Sort y with  $y > x$ ) when it is necessary. For instance, imagine you have some function that takes as input ( $x : \text{Sort } 1$ ) but that you would like to apply to a Prop (i.e.  $y : \text{Sort } 0$ ) then you can apply it to a lifted version of y which will live in Sort 1.

More examples can be found at the bottom of [this page](#).