Mathematical project in L∃∀N

An innocent mathematicians guide to lean

Andreas Bøgh Poulsen, studienummer: 201805425

February 19, 2023

Contents

Abstract

Introduction

The following is a project, in which I try to learn how to do formalized mathematics, using Lean as my proof checker. This document is a report on my learnings, and is intended as a resource for other mathematicians, who may wish to learn about Lean.

1 Martin-Löf dependent type theory

Dependent type theory is a logical theory, comparable to first order logic. Similarly to how we usually think we do mathematics in first order logic with ZFC set theory on top, we can translate our mathematical theories into other logical theories. In this chapter I'll give a taste of how dependent type theory works as a formal system. If you're only interested in learning Lean, feel free to skip this section.

1.1 Inference rules

A Inference rule is on the form

$$\frac{\Gamma \vdash P \qquad \Gamma \vdash Q}{\Gamma \vdash P \wedge Q} \ \wedge\text{-intro}$$

which is read as follows: if we, in a context Γ, can prove P and in the same environment can prove Q, then we can prove $P \wedge Q$ in the context Γ.

The defining feature of *t*ype theory is, that every element has a type. Thus the above is meaningless, as P and Q have no type. Compare this to ZFC, where everything is either a proposition from first order logic, or a set. This leads to weird statements like $0 \in 1$, which is well-posed since everything is a set, but does not carry a meaning in our "usual" way of doing mathematics. Type theory asks that every element has a type. This is particularly helpful when doing computerized proofs, as it helps the proof-checker catch weird statements like $0 \in 1$. Since 1 has the type of a natural number and the type of a set, it can give an error, instead of silently trying to prove what may well have been a typo.

In type theory the above rewrite rule would look like this:

$$\frac{\Gamma \vdash P : Prop \qquad \Gamma \vdash Q : Prop}{\Gamma \vdash P \wedge Q : Prop} \ \wedge\text{-intro}$$

Everything is read the same, except $P : Prop$ is read "P has type *Prop*". *Prop* is the type of propositions. I will not spend too much time going through every single single inference rule. I will, however, introduce the defining features of dependent type theory: dependent types, and show how they are used.

1.1 · Definition. Type theory has four different *judgements*.

1. $\Gamma \vdash A$ type says A is a well-formed type in context Γ.
2. $\Gamma \vdash A \doteq B$ type says A and B are judgementally equal types in context Γ.
3. $\Gamma \vdash a : A$ says a is an element of type A in context Γ.
4. $\Gamma \vdash a \doteq b$ type says a and b both have type A and are judgementally equal.

As we would expect, there are axioms making this an equivalence relation:

$$\frac{\Gamma \vdash a : A}{\Gamma \vdash a \doteq a : A} \qquad \frac{\Gamma \vdash a \doteq b : A}{\Gamma \vdash b \doteq a : A} \qquad \frac{\Gamma \vdash a \doteq b : A \quad \Gamma \vdash b \doteq c : A}{\Gamma \vdash a \doteq c : A}$$

and similarly for types. There is also a rule stating that you can substitute judgementally equal elements anywhere.

Judgemental equality is actually a very strong equality, and many objects we usually consider equal, cannot be proven judgementally equal. Later we'll introduce a weaker equality, that captures better our usual understanding of equality. Stay tuned, the formulation may surprise you.

We need to introduce dependent types as well as functions, before we can get going.

1.2 · Definition. A *dependent type* is a type of the form $\Gamma, x : A \vdash B(x)$ type with a rule letting us assume elements of that type:

$$\frac{\Gamma \vdash A \text{ type}}{\Gamma a : A \vdash a : A}$$

A *section* of a dependent type $B(x)$ is an element $\Gamma, x : A \vdash b(x) : B(x)$.

Note that for different $x : A$ in the context, $B(x)$ may be different type. Using dependent types we can introduce functions:

1.3 · Definition. A *function type* is the type of sections of a dependent type $B(x)$, given by the following introduction rules:

$$\frac{\Gamma, x : A \vdash B(x) \text{ type}}{\Gamma \vdash \Pi_{x:A} B(x) \text{ type}} \qquad \frac{\Gamma, x : A \vdash b(x) : B(x)}{\Gamma \vdash \lambda x, b(x) : \Pi_{x:A} B(x)}$$

and has the following evaluation rules:

$$\frac{\Gamma \vdash f : \Pi_{x:A} B(x)}{\Gamma, x : A \vdash f(x) : B(x)} \qquad \frac{\Gamma, x : A \vdash b(x) : B(x)}{\Gamma, x : A \vdash (\lambda x, b(x))(x) \doteq b(x) : B(x)}$$

Remark Not all types are dependent. If $B(x)$ is independent of x we will just write B and functions as $A \to B$.

1.2 Logic in type theory

We now have the building blocks to start formulating usual logic in dependent type theory. The basic idea is to interpret types as propositions. A proof of a proposition corresponds to an element of a type. Thus a false proposition is a type without any elements, and a true proposition is a type with at leat one element. In this interpretation, implication simply becomes a function. $f : A \to B$ says "f takes an element of A and produces an element of B" or as propositions "f takes a proof of A and produces a proof of B", which is exactly what an implication does.

$$\frac{}{\vdash \emptyset} \qquad \frac{}{\vdash ind_{\emptyset} : \Pi_{x:\emptyset} P(x)}$$

The rule $ind_{\emptyset}$ states, that we can prove everything from false.

where $ind_{\emptyset}$ states that everything is true about elements of the empty type. Thus we can interpret something being false $\neg A$ as the type $A \to \emptyset$. We can then prove the statement $(A \implies B) \implies (\neg B \implies \neg A)$. In type theory, this translates to $(A \to B) \to ((B \to \emptyset) \to (A \to \emptyset))$. The construction is as follows:

$$\frac{\frac{\frac{\frac{\frac{\Gamma \vdash A \text{ type}}{\Gamma, a : A \vdash a : A} \quad \frac{\frac{\frac{\Gamma \vdash A \text{ type}}{\Gamma, a : A \vdash a : A} \quad \Gamma \vdash B \text{ type}}{\Gamma \vdash A \to B \text{ type}}}{\Gamma, h : A \to B \vdash h : A \to B}}{\Gamma, a : A, h : A \to B \vdash h(a) : B} \quad \frac{\frac{\frac{\Gamma \vdash B \text{ type}}{\Gamma, b : B \vdash b : B} \quad \frac{}{\vdash \emptyset \text{ type}}}{\Gamma \vdash B \to \emptyset \text{ type}}}{\Gamma, f : B \to \emptyset \vdash f : B \to \emptyset}}{\Gamma, a : A, f : B \to \emptyset, h : A \to B \vdash f(h(a)) : \emptyset}}{\Gamma, f : B \to \emptyset, h : A \to B \vdash \lambda a, f(h(a)) : A \to \emptyset}}{\Gamma, h : A \to B \vdash \lambda f, \lambda a, f(h(a)) : (B \to \emptyset) \to (A \to \emptyset)}}{\Gamma \vdash \lambda h, \lambda f, \lambda a, f(h(a)) : (A \to B) \to ((B \to \emptyset) \to (A \to \emptyset))}$$

You'll note that we didn't use $ind_{\emptyset}$ in the constrution. Indeed, this is a special case of the more general formula $(A \to B) \to ((B \to C) \to (A \to C))$, which we get simply by composing functions. We'll denote $f \circ g := \lambda x, f(g(x))$ and refer to the above proof tree for its construction.

So how do we actually use $ind_\emptyset$? Well, we can't prove much right now, but if we introduce *or*:

$$\frac{\Gamma \vdash A \text{ type} \quad \Gamma \vdash B \text{ type}}{\Gamma \vdash A \vee B \text{ type}} \qquad \frac{\Gamma \vdash a : A \quad \Gamma \vdash B \text{ type}}{\Gamma \vdash \iota_1 : A \to A \vee B} \qquad \frac{\Gamma \vdash A \text{ type} \quad \Gamma \vdash b : B}{\Gamma \vdash \iota_2 : B \to A \vee B}$$

$$\frac{\Gamma \vdash A \text{ type} \quad \Gamma \vdash B \text{ type}}{\Gamma \vdash ind_\vee : (\Pi_{a:A} P(\iota_1(a))) \to (\Pi_{b:B} P(\iota_2(b))) \to (\Pi_{z:A \vee B} P(z))}$$

$$\frac{\Gamma \vdash a : A \quad \Gamma \vdash l : \Pi_{a:A} P(a) \quad \Gamma \vdash r : \Pi_{b:B} P(b)}{\Gamma \vdash ind_\vee(l, r, \iota_1(a)) \doteq l(a) : P(a)}$$

$$\frac{\Gamma \vdash b : B \quad \Gamma \vdash l : \Pi_{a:A} P(a) \quad \Gamma \vdash r : \Pi_{b:B} P(b)}{\Gamma \vdash ind_\vee(l, r, \iota_2(b)) \doteq r(b) : P(b)}$$

we can prove the following: $\neg A \to (A \vee B) \to B$.

$$\frac{\Gamma \vdash A \text{ type} \quad \Gamma \vdash B \text{ type}}{\Gamma \vdash \lambda h, \lambda z, ind_\vee(ind_\emptyset \circ h, id, z) : (A \to \emptyset) \to (A \vee B) \to B}$$

Okay, that was quite a mouthful. Let's work through the rules for $\vee$ in order. First, assuming two types A and B, we can form the disjunction $A \vee B$. We have two rules for forming elements of $A \vee B$, namely ι_1 and ι_2 which take an element of A, resp. B and forms an element of $A \vee B$. Next line, we have a way to use a disjunction. Given a proof of P assuming A and a proof of P assuming B, we get proof of P assuming $A \vee B$. The final two lines state, that $ind_\vee$ behaves the way we expect it to.

Using these, the proof if the assertion above becomes

$$\dfrac{\dfrac{\dfrac{\dfrac{\Gamma \vdash A \text{ type} \quad \Gamma \vdash B \text{ type}}{\Gamma \vdash A \vee B \text{ type}}}{\Gamma, z : A \vee B \vdash z : A \vee B} \quad \dfrac{\dfrac{\dfrac{\Gamma \vdash A \text{ type} \quad \dfrac{}{\vdash \emptyset \text{ type}}}{\Gamma \vdash A \to \emptyset \text{ type}}}{\Gamma, h : A \to \emptyset \vdash h : A} \quad \dfrac{}{\vdash ind_\emptyset : \ldots}}{\Gamma \vdash ind_\emptyset \circ h : A \to B} \quad \dfrac{\Gamma \vdash A \text{ type} \quad \Gamma \vdash B \text{ type}}{\Gamma \vdash ind_\vee : \ldots}}{\dfrac{\Gamma, h : A \to \emptyset, z : A \vee B \vdash ind_\vee(ind_\emptyset \circ h, id, z) : B}{\Gamma, h : A \to \emptyset \vdash \lambda z, ind_\vee(ind_\emptyset \circ h, id, z) : (A \vee B) \to B}}}{\Gamma \vdash \lambda h, \lambda z, ind_\vee(ind_\emptyset \circ h, id, z) : (A \to \emptyset) \to (A \vee B) \to B}$$

I have omitted some types to make the tree fit the page, but the crux of the argument is, that from an implication $A \to \emptyset$ and a proof of A, we can use $ind_\emptyset$ to prove B. Thus we have a function $(A \to \emptyset) \to B$, which we can use, together with $id : B \to B$ to prove B from a $A \vee B$.

Okay, so we have negation, implication and disjunction. I encourage you to imagine how conjunction would be defined. But what about quantors? We'll postpone the existential quantor until later, as it's formulation is quite subtle, but universal quantification is surprisingly straightforward. $\forall x.P(x)$ states that for every x x, we get a proof of $P(x)$. That sounds like a function to me. And indeed, we simply define $\forall := \Pi$.

This may be surprising, but it actually highlights a strength of dependent type theory as a logical framework: everything, even proofs, is just elements of types. The disjunction, as defined above, is also known as the coproduct in functional programming languages. In the next section, we'll take full advantage of this idea.

1.3 The natural numbers

So far we've only thought about propositions. Let's introduce to natural numbers, as an example of something non-propositional.

$$\frac{}{\vdash \mathbb{N} \text{ type}} \qquad \frac{}{\vdash 0_{\mathbb{N}} : \mathbb{N}} \qquad \frac{}{\vdash succ_{\mathbb{N}} : \mathbb{N} \to \mathbb{N}}$$

$$\frac{\Gamma, n : \mathbb{N} \vdash P(n) \text{ type} \qquad \Gamma \vdash p_0 : P(0_{\mathbb{N}}) \qquad \Gamma \vdash p_s : \Pi_{n:\mathbb{N}}(P(n) \to P(succ_{\mathbb{N}}(n)))}{\Gamma \vdash ind_{\mathbb{N}}(p_0, p_s) : \Pi_{n:\mathbb{N}} P(n)}$$

$$\frac{\Gamma, n : \mathbb{N} \vdash P(n) \text{ type} \qquad \Gamma \vdash p_0 : P(0_{\mathbb{N}}) \qquad \Gamma \vdash p_s : \Pi_{n:\mathbb{N}}(P(n) \to P(succ_{\mathbb{N}}(n)))}{\Gamma \vdash ind_{\mathbb{N}}(p_0, p_s, 0_{\mathbb{N}}) \doteq p_0 : P(0_{\mathbb{N}})}$$

$$\frac{\Gamma, n : \mathbb{N} \vdash P(n) \text{ type} \qquad \Gamma \vdash p_0 : P(0_{\mathbb{N}}) \qquad \Gamma \vdash p_s : \Pi_{n:\mathbb{N}}(P(n) \to P(succ_{\mathbb{N}}(n)))}{\Gamma \vdash ind_{\mathbb{N}}(p_0, p_s, succ_{\mathbb{N}}(n)) \doteq p_s(n, ind_{\mathbb{N}}(p_0, p_s, n)) : P(succ_{\mathbb{N}}(n))}$$

The first three rules govern the construction of natural numbers, and the next rule is the induction rule. If we for a moment assume P is a predicate, it reads "Given a predicate P, a proof of $P(0)$ and proof of $P(n) \implies P(succ(n))$ we get a proof of $\forall n : P(n)$." The two final rules simply state, that induction behaves as we expect.

All these inference rules are quite heavy. Let's introduce some lighter notation:

```
type ℕ :=
| zero_ℕ : ℕ
| succ_ℕ : ℕ → ℕ
```

Everything in the inference rules is derivable from this definition. We can similarly define

```
type A∨B :=
| ι₁ : A → A∨B
| ι₂ : B → A∨B
```

Let's define addition and prove some identities. We would like addition to respect the following specification:

$$\begin{gathered} add_{\mathbb{N}}(0,n) \doteq n \\ add_{\mathbb{N}}(m, succ_{\mathbb{N}}(n)) \doteq succ(add_{\mathbb{N}}(m,n)) \end{gathered}$$

and we would like to do it using the induction rule on $\mathbb{N}$. Remember $ind_{\mathbb{N}}(p_0, p_s)$ has type $\Pi_{n:\mathbb{N}} P(n)$ and addition needs to have type $\mathbb{N} \to (\mathbb{N} \to \mathbb{N})$. Thus P needs to have type $\mathbb{N} \to \mathbb{N}$. The idea is that $ind_{\mathbb{N}}(p_0, p_s, n)$ should produce a function adding n to a number. Then $ind_{\mathbb{N}}(p_0, p_s, n)(m)$ computes $n+m$.

First, let's define $p_0 := id : \mathbb{N} \to \mathbb{N}$. This is a function taking a number and adding 0 to it. Then we need to define $p_s : \mathbb{N} \to (\mathbb{N} \to \mathbb{N}) \to (\mathbb{N} \to \mathbb{N})$, that is, given a number n and a function adding n to a number, return a function adding $n+1$ to a number. This is simply $p_s(n, f) := succ_{\mathbb{N}} \circ f$. Thus

$$add := \lambda m, \lambda n, ind_{\mathbb{N}}(id, \lambda f, succ_{\mathbb{N}} \circ f, n)(m) : \mathbb{N} \to \mathbb{N} \to \mathbb{N}.$$

We can see that it satisfies our specification:

$$\begin{aligned} add_{\mathbb{N}}(0,n) &\doteq \lambda m, \lambda n, ind_{\mathbb{N}}(id, \lambda f, succ_{\mathbb{N}} \circ f, 0_{\mathbb{N}})(m) \\ &\doteq id(0_{\mathbb{N}}) \\ &\doteq 0_{\mathbb{N}} \end{aligned}$$

$$\begin{aligned} add_{\mathbb{N}}(m, succ_{\mathbb{N}}(n)) &\doteq \lambda m, \lambda n, ind_{\mathbb{N}}(id, \lambda f, succ_{\mathbb{N}} \circ f, succ_{\mathbb{N}}(n))(m) \\ &\doteq (\lambda f, succ_{\mathbb{N}} \circ f)(n, ind_{\mathbb{N}}(id, \lambda f, succ_{\mathbb{N}} \circ f), n)(m) \end{aligned}$$

or written as a proof tree

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\vdash ind_{\mathbb{N}} : \ldots}}{n : \mathbb{N}, m : \mathbb{N} \vdash ind_{\mathbb{N}}(id, \lambda f, succ_{\mathbb{N}} \circ f, n)(m) : \mathbb{N}}}{n : \mathbb{N} \vdash \lambda m, ind_{\mathbb{N}}(id, \lambda f, succ_{\mathbb{N}} \circ f, n)(m) : \mathbb{N} \to \mathbb{N}}}{\vdash \lambda n, \lambda m, ind_{\mathbb{N}}(id, \lambda f, succ_{\mathbb{N}} \circ f, n)(m) : \mathbb{N} \to \mathbb{N} \to \mathbb{N}}$$

2 Mathematics in type theory and Lean

3 Gröbner bases as an extended example