Introduction to Proof Assistants

Using Lean4

Christian Benedict Smit

TECHNISCHE UNIVERSITÄT DARMSTADT

2025-07-08

My Introduction

- 2017: B.Sc. Biology - Ruhr University Bochum
- 2021: B.Sc. Computer Science - TU Dortmund
- 2025: M.Sc. Computer Science - TU Dortmund
  - Masters Thesis: “Congruence Closure in the presence of dependent types”
  - Implementations in Rocq (formerly known as Coq)
- 2025: Start of my Ph.D. in Computer Science
  - TU Darmstadt
  - Software Technology Group of Mira Mezini
- For my Ph.D. I started using Lean4

What is Proof Assistants

- Programs to write proofs
- Proofs are programs in a typed purely functional programming language
  - Functional meaning the main abstraction are functions
  - Purely meaning those functions are mathematical functions
- Using constructive mathematics with intuitionistic logic
- Underlying is the Calculus of Inductive Constructions (CIC)
  - A typed Lambda Calculus
  - Types can also contain terms
- A small trusted kernel does the type checking

Motivation to use Proof Assistants

- Preventing errors in proofs
  - 1852: Francis Guthrie proposes the Four Color Theorem
  - 1879: Alfred Kempe proposes a proof for the Four Color Theorem
  - 1890: Percy Heawood shows, that Kempe's proof has faults
- Proof automation
  - 1976: Kenneth Appel and Wolfgang Haken proof the Four Color Theorem computer aided
  - 2005: Georges Gonthier formalizes a proof for the Four Color Theorem in Coq
- Preventing Bugs in Software

My Masters Thesis

Rocq

```
Goal forall (A: Type) (a a' b: A), (a, b) = (a', b) -> a = a'.
Proof.
    congruence.
Qed.
```

Rocq

```
Inductive Vector A : nat -> Type :=
| nil : Vector A 0
| cons : forall (h: A) {n: nat}, Vector A n -> Vector A (S n).
```

Rocq

```
Goal forall (A: Type) (n: nat) (h: A) (t t': Vector A n),
  cons h t = cons h t' -> t = t'.
Proof.
  congruence.
```

Lambda Calculus

untyped

Let M, N be other lambda terms

x : Variable

$(\lambda x.N)$: Lambda Abstraction

(MN) : Application

typed

$x : T$: Variable

$(\lambda x : T.N)$: Lambda Abstraction

(MN) : Application

reduction

$$((\lambda x.M)N) \underset{\beta}{\rightarrow} (M[x/N])$$

typing

$$\frac{}{\Gamma, x : T \vdash x : T}\textsc{Var}$$

$$\frac{\Gamma, x : T_1 \vdash t_2 : T_2}{\Gamma \vdash \lambda x : T_1.t_2 : T_1 \rightarrow T_2}\textsc{Abs}$$

$$\frac{\Gamma \vdash t_1 : T_2 \rightarrow T_{12} \quad \Gamma \vdash t_2 : T_2}{\Gamma \vdash t_1\, t_2 : T_{12}}\textsc{App}$$

Curry-Howard correspondence

Types $\approx$ Propositions

Terms $\approx$ Proofs

Let Prop be the Type of propositions

logic	programming	explanation
$A \wedge B$	$A \times B$	product type
$A \vee B$	$A \oplus B$	tagged sum type
$A \Rightarrow B$	$A \to B$ or $\forall \_ : A, B$	function type
$\forall x : T, P$	$\forall x : T, P$	dependent function type
$\exists x : T, P$	$(x : T, P\, x)$	dependent pair type
True	Unit also called True	Type with the one element $()$
False	Void also called False	Type with no elements
$\neg A$	$A \to \mathrm{False}$	function to False

Live Coding

Now lets code

Further interesting sources

Cheat sheet:

- https://leanprover-community.github.io/papers/lean-tactics.pdf

The Number Game:

  - https://adam.math.hhu.de/
  - Good for beginners

Glimps of Lean:

  - https://github.com/PatrickMassot/GlimpseOfLean
  - Shows different arias of Mathlib

Fermats Last Theorem in Lean:

  - https://leanprover-community.github.io/blog/posts/FLT-announcement/
  - Everyone can contribute

Thanks

Thank you for listening :-)

Defining new data types

We can define own data types with inductive definitions

```lean
inductive Nat : Type where
  | zero : Nat
  | succ : Nat → Nat
```

And we can define our own proposition types

```lean
inductive XOr (A B : Prop) : Prop where
  | left_not_right : A → ¬B → XOr A B
  | right_not_left : ¬A → B → XOr A B
```

Abstract structures with type classes

Define properties of structures:

Lean

```lean
class JoinSemiLattice (α: Type) : Type where
  join : α → α → α
  assoc : join a (join b c) = join (join a b) c
  comm : join a b = join b a
  idem : join a a = a
```

Abstract structures with type classes

Give concrete implementations for those properties:

```lean
instance nat_join_semi_lattice : JoinSemiLattice Nat where
  join := max
  assoc := by simp
  comm := by apply?
  idem := by simp
```