Introduction to Lean 4

MATH230

School of Mathematics and Statistics
University of Canterbury

Outline

Curry-Howard Correspondence

$$\Sigma \vdash_{\mathsf{I}} \alpha \iff \Sigma \vdash_{\mathsf{STT}} \alpha$$

Each natural deduction of α from hypotheses Σ is equivalent to a program of type α in the context Σ.

This suggests that we should be able to develop a programming language for which:

- Propositions are particular types in that language,
- The programs inhabitating those types are the proofs of the corresponding proposition,
- Proofs can be validated by type checking, and
- Proof authoring can be helped by higher-order programs.

IPE: Integrated Proving Environments

Lean is a functional programming language and an interactive theorem prover. It can be used to write general purpose computer programs and as an assistant in the process of authoring and verifying proofs about mathematics and software. This open source project was launched by Leonardo de Moura at Microsoft Research in 2013. Lean 4 is the latest version and is maintained by de Moura and others at the Lean Focussed Research Institute.

Lean is not the only language to implement the theoretical ideas that we have discussed throughout this course. Other languages include Agda, Idris, and Rocq. For the purposes of this course we will only be using Lean.

Getting Started

It is simplest to run Lean through the editor VS Code. Following the instructions at this link shows you how to do this. There are plugins for other editors, if you're that way inclined. However, if you're new to programming, then it is recommended you stick to VSCode. This is the setup available in the computer labs of Jack Erskine.

These lectures were prepared with reference to the free textbook Theorem Proving in Lean 4. This can be referred to for more details.

Example

$$\cfrac{\cfrac{\cfrac{\overline{A}^{\,1} \qquad A \to B}{B}\ \text{MP} \qquad B \to C}{C}\ \text{MP}}{A \to C}\ \to I, 1$$

$$\cfrac{\cfrac{\cfrac{\overline{a : A}^{\,1} \qquad f : A \to B}{f\ a : B}\ \text{app} \qquad g : B \to C}{g\ (f\ a) : C}\ \text{app}}{\lambda x.\ g\ (f\ x) : A \to C}\ \lambda, 1$$

In the end it is sufficient to provide the proof-term:

$$\lambda x.\ g\ (f\ x) : A \to C$$

Lean Demo

Before we talk about the syntax of Lean4; let's just see how it works with two familiar examples:

$$A \to B, B \to C \ \vdash \ A \to C$$

```
fun a => g (f a)
```

$$A \land B \to C \ \vdash \ A \to B \to C$$

```
fun a => fun b => f (And.intro a b)
```

de Moura's Stone

PL	$\boldsymbol{\lambda}$	**L∃∀N 4**
$\wedge I$	(p, q)	And.intro p q
$\wedge E_l$	fst t	And.left t
$\wedge E_r$	snd t	And.right t
$\rightarrow I$	$\lambda\ p : P.$	$\lambda\ p : P =>$
$\rightarrow E$	$(f\ \ t)$	$(f\ \ t)$
$\vee I_l$	inl p	Or.intro_left <right-disj> p
$\vee I_r$	inr p	Or.intro_right <left-disj> p
$\vee E$	cases t f g	Or.elim t f g

Table: Syntax of logic, λ-calculus, and L∃∀N 4

Examples from the previous page show that we have all but written Lean4 programs already. There are just minor syntax changes between the Simple Type Theory we have studied and the syntax of Lean4.

Keywords: Variables

Lean, indeed all programming languages, have a number of keywords used to structure programs. For our purposes of theorem proving, we will not need to know all of the keywords of Lean.

One can declare type, or prop, variables globally or locally. They can be declared globally as follows:

```
variable (P Q R : Prop)
```

This puts propositional variables P, Q, and R into the context of the module you're writing. As they are propositions, Lean knows that can have the logical operations introduced above applied to them.

These variables can also be introduced locally into the definition/theorem one is writing. We will see how to do this below.

Keywords: Thereom

Theorem statements have the following syntax in Lean4. Note the similarities with our usual sequent notation.

```
theorem <name> <hypotheses> : <goal> := <proof−term>
```

Some things to note:

- Each hypothesis should be written with its own parentheses.
- Use whitespace, not commas, to separate hypotheses.
- Goal is the Type/Prop to be proved.
- Proof-term is is the proof of that Prop.
- Theorems are just functions. One can use def instead.
- One can opt for a nameless theorem with the "example" keyword.

```
example <hypotheses> : <goal> := <proof−term>
```

Lean Infoview

Lean's infoview is one of its key features. Along with the editor one writes the proof in, Lean provides another infoview to display a lot of information regarding the current proof. This is indespensible when proofs start to get even moderately long.

Example

We will work through the following examples to get a taste of theorem proving in Lean4:

$\vdash\ P \wedge Q \to Q \wedge P$

$\vdash\ P \vee Q \to Q \vee P$

$\vdash\ P \to P$ [I Combinator]

$\vdash\ P \to (Q \to P)$ [K Combinator]

$\vdash\ (P \to Q \to R) \to ((P \to Q) \to P \to R)$ [S Combinator]

$P \to Q, \neg Q\ \vdash \neg P$ [Modus Tollens]

$P \to Q\ \vdash\ \neg Q \to \neg P$ [Intuitionistic Contrapositive]

$(P \wedge Q) \wedge R\ \vdash\ P \wedge (Q \wedge R)$

$(P \vee Q) \vee R\ \vdash\ P \vee (Q \vee R)$

$\vdash\ (P \wedge Q \to R) \leftrightarrow (P \to Q \to R)$

Tactics

Proof-verification alone is all well and good. However, Lean4 (and the other proof assistants) have a number of built in metaprograms (tactics) to help in the authoring of proofs.

Moreover, because Lean4 is a general purpose programming language one can write new tactics in Lean to add further simplifcations to the process of writing proof terms.

Tactics proofs do not write proof-terms explicitly, but use higher-order programs to help write the required proof-term. Remember that proof-terms are the object of interest - they are the certificate that is verified to know when a proof is complete. Even when we write tactic proofs, we still generate a proof-term.

Tactic: intro

The tactic “intro” can be applied to simplfy a goal of the form $\Sigma \vdash \alpha \to \beta$ to the updated goal $\Sigma, \alpha \vdash \beta$. This puts the antedecent in the context and takes care of the implication introduction aka function abstraction we would normally do explictly at such a step.

This tactic takes argument(s) which are used as names for the terms of the corresponding proposition/type introduced.

If no arguments are passed to intro, then the term is anonymous. As such it is more difficult to make use of that hypothesis later in the proof. For this reason we will always name the terms inhabiting the hypotheses we introduce.

This tactic **infers** the type of the term variable introduced from the goal. If the goal is not an implication, then the Lean will throw an error.

Tactic: apply

The "apply" tactic takes a theorem or constructor of the form $\Sigma \vdash \alpha$ and matches the current goal with the conclusion α in the theorem applied. This tactic updates the goal to a (list of) new goal(s) depending on the hypotheses used in the theorem.

For example, And.intro has two arguments $a : A$ and $b : B$ and the conclusion $A \wedge B$. If the goal of the current proof-state is a conjunction $P \wedge Q$, then using "apply And.intro" will update the proof-state to have two goals: one asking for a proof of P and another asking for a proof of Q. All of the $\wedge$ introduction details are taken care of behind the scenes.

Tactic: apply

We can make use of apply with a hypothesis of the form $t : P \vee Q$. If the current proof-state is of the form $\Sigma, t : P \vee Q \ \vdash \gamma$, then we can use "apply Or.elim t" to split the proof into two cases: $\Sigma \vdash P \to \gamma$ and $\Sigma \vdash Q \to \gamma$.

Or.elim takes three arguments (i) $t : P \vee Q$ (ii) $f : P \to \gamma$, and (iii) $g : Q \to \gamma$. Using apply with Or.elim t then reduces the problem to determining the other two arguments f, g respectively.

Tactic Summary

Tactic	Summary
by	Opens tactic mode.
intro	introduces named variable(s) with inferred type
apply	calls a theorem to update goal
have	creates an intermediate term (sub-proof)
exact	closes a goal by providing a term of required type.

These tactics are sufficient for the minimal logic proofs in the labs and assignment.

Theorem Proving in Lean 4 chapters 3 and 5 have more detail on proving propositional logic theorems in Lean 4.

Predicates

Recall from earlier in the course that we defined predicates on $\mathbb{N}$ to be functions:

$$P : \mathbb{N} \to \mathsf{Prop}$$

Specifically, for each natural number $n : \mathbb{N}$ we get a proposition

$$P\ \ n : \mathsf{Prop}$$

Adding the following puts a type variable A, term variables inhabiting that type, and unary predicates F, G, H into the context of the Lean file.

```
variable (A : Type)
variable (a b c t u v : A)
variable (F G H : A -> Prop)
```

For now we will talk about first order logic on arbitrary types and return later to talk about specific types like $\mathbb{N}$.

BHK

Although we did not cover the specific type constructors corresponding to the quantifiers $\forall$ and $\exists$ we can use the BHK to understand how to prove quantified claims.

$\forall$	a proof of $\forall x\ P(x)$ is an algorithm that, applied to *any* object t returns a proof of $P(t)$.
$\exists$	to prove $\exists x\ P(x)$ one must construct an object t and a proof of $P(t)$.

One may also refer to Simon Thompson, or Theorem Proving in Lean 4 for more details.

Universal Quantifier

The constructor $\forall$ builds a proposition from (i) a type α (ii) a variable $x : \alpha$ of that type, and (iii) a unary predicate $P : \alpha \to \mathrm{Prop}$ over that type.

$$f \ : \ \forall x : \alpha, \ P \ x$$

Following the BHK, each term f of this type is to be thought of as a function which takes in any $a : \alpha$ and returns a proof $p : (P \ a)$ that a satifies the predicate P.

Therefore the type destructor for $\forall$ is simply function application.

To construct a term of this type we must write such a function.

Existential Quantifier

The constructor $\exists$ builds a proposition from (i) a type α (ii) a variable $x : \alpha$ of that type, and (iii) a unary predicate $P : \alpha \to \text{Prop}$ over that type.

$$t \;:\; \exists x : \alpha,\; P\; x$$

Following the BHK, each term t of this type is to be thought of as a pair (a, p) where $a : \alpha$ and $p : (P\; a)$ is a proof that a satisfies the predicate P. In this context we refer to a as the witness to this existential claim and p the proof that a is the witness.

To construct a term of this type requires both of these pieces of information.

We can unpack a term t into the witness and the proof the witness has the property.

Universal Quantifier and Lean

If $f : \forall x : \alpha, P\ x$ is a hypothesis in the context and $a : \alpha$, then function application $f\ a$ will return a term of type $P\ a$ i.e. a proof that a satifies the predicate p.

If the goal $\Sigma \vdash\ \forall x : \alpha, P\ x$ is a universally quantified statement, then one can use the intro tactic to construct the required function. This time the name passed to intro will the name given to the arbtirary term $a : \alpha$.

Existential Quantifer and Lean

If $t : \forall x : \alpha, P\ x$ is a hypothesis in the context then one can use the tactic call:

```
rcases t with a p
```

This gives the names a to the term satifying P and p to the proof that a satifies P.

If the goal $\Sigma \vdash\ \exists x : \alpha, P\ x$ is an existential statement, then one can use the term builder Exists.intro as follows to build the required goal:

```
Exists.intro a p
```

Again, a is the name of the term statisfying the predicate and p is the proof that a satifies the predicate.

Example

We will work through the following examples to get a taste of theorem proving in Lean4:

$$\forall x\ Fx \to Gx,\ \ Fa \vdash\ G\ a$$

$$\forall x\ Fx \to Gx,\ \forall x\ Fx \vdash\ \forall x\ Gx$$

$$\forall x\ Fx \to Gx,\ \forall x\ Gx \to Hx \vdash\ \forall x\ Fx \to Hx$$

$$\forall x\ Fx \to Gx,\ \neg Ga \vdash\ \exists x\ \neg Fx$$

$$\forall x\ Fx \to Gx,\ \exists x\ F\ x \vdash\ \exists x\ G\ x$$

$$(\exists x\ F\ x) \lor (\exists x\ G\ x) \vdash\ \exists x\ (F\ x \lor G\ x)$$

$$\neg\exists x\ F\ x \vdash\ \forall x\ \neg F\ x$$

Further Reading

These slides were prepared with the aid of the following references. These should be consulted for further detail on the topics.

Type Theory and Functional Programming, Simon Thompson

Theorem Proving in Lean 4

Each of these are freely available on the world wide web.