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.

Lean 4

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

$$\dfrac{\dfrac{\dfrac{\overline{A}^{\,1} \quad A \to B}{B}\,\mathrm{MP} \qquad B \to C}{C}\,\mathrm{MP}}{A \to C}\,\to I, 1$$

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

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)
```

$$A \wedge B \to C \vdash A \to B \to C$$

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

PL	λ	L$\exists\forall$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$\exists\forall$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 following 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 Summary

Lean 4 Tactics

Tactic	Summary	Natura
by	Opens tactic mode.	
intro(s)	Either (i) implication introduction, or (ii) $\forall$ introduction.	Let n b
exact	This term has the same type as the goal.	... as r
apply		
have	Introduces a local variable, or intermediate step.	So far
rfl	Closes goals that are (up to normalization) definitionally equal.	... by d
rw	Substitutes equals for equals.	
induction	Wraps base case and induction step into $\forall$ proof.	By indu
calc	Line-by-line calculation showing $a = \cdots = b$.	Algebra

Further Reading

This lecture was prepared with the aid of the following references. These should be consulted for further detail on the topics.