Lecture 3: From Natural Deduction to Proofs in Lean

Stefano M. Nicoletti

1 Lecture 3: From Natural Deduction to Proofs in Lean

1.1 Aim of the lecture

In the previous lecture, we learned to read a proof state and used tactics such as `intro`, `apply`, `have`, `exact`, and `cases`. We now connect those commands to the rules of natural deduction. Tactics are not arbitrary moves: they build and use proofs according to the logical structure of propositions.

The lecture has five aims:

- revisit the idea of type inhabitation;
- introduce the Curry-Howard correspondence;
- read introduction and elimination rules in natural deduction;
- recognize those rules in Lean proofs;

1.2 Types and inhabitants

A type describes a class of possible objects. A type is *inhabited* when there is at least one term of that type. If `t` has type `T`, we write `t : T` and say that `t` inhabits `T`.

The idea is not limited to Lean. The informal type "moon of Jupiter" is inhabited because Io is a moon of Jupiter. Io is a concrete object satisfying the description expressed by the type (Source: NASA Science, Io: Facts).

```
#check 4          -- 4 : Nat
#check True       -- True : Prop
#check True.intro -- True.intro : True
```

`Nat` is inhabited, for example, by `4`. `True` is also inhabited: `True.intro` has exactly type `True`. This observation leads from the general relation between terms and types to the relation between proofs and propositions.

1.3 The Curry-Howard correspondence

Under the Curry-Howard correspondence, propositions and types are two ways of describing the same structure. A proposition specifies what must be proved; the corresponding type specifies what kind of term must be built. A proof of the proposition is a term inhabiting that type (Source: Wadler, Propositions as Types).

Logic	Type theory
proposition	type
proof	term
`A → B`	function from proofs of `A` to proofs of `B`
introduction of →	function construction
elimination of →	function application

Checking a proof therefore means checking that the constructed term has the declared type. A derivation does not merely certify that a proposition is true: it specifies how to construct its proof (Source: Theorem Proving in Lean 4, Propositions and Proofs).

1.4 Natural deduction and the proof state

Natural deduction presents a proof as a sequence of local steps governed by rules. Each rule shows how to obtain a conclusion from premises. Introduction rules explain how to construct a proof with a particular main connective; elimination rules explain how to use an available proof containing that connective (Source: Logic and Proof, Natural Deduction for Propositional Logic).

A judgment has the form $\Gamma \vdash A$. Here Γ is the context of currently available assumptions and A is the conclusion to derive. Lean's proof state has the same organization: the context appears above the turnstile and the goal after it.

Some rules introduce temporary assumptions. To prove `A → B`, for example, we temporarily assume `A` and construct `B`. Once the implication is complete, the assumption has been incorporated into a function from proofs of `A` to proofs of `B`.

1.5 Implication

1.5.1 Implication introduction, '→I'

$$\dfrac{\dfrac{}{A}\,1}{\begin{array}{c}\vdots\\ B\end{array}}\;\dfrac{}{A \to B}\,1\;\to$$

Implication introduction, →I

To prove `A → B`, we introduce an assumption of type `A` and construct a proof of `B`. In Lean, `intro` performs precisely this step.

We want to prove that if it rains and it is cold, then it rains. We introduce the assumption that it rains and is cold; we obtain that it rains by taking the left component; we use exactly that fact.

```
theorem lecture03_imp_intro
    (Rains IsCold : Prop) :
    Rains ∧ IsCold → Rains := by
  intro hRainsAndIsCold
  have hRains := hRainsAndIsCold.left
  exact hRains
```

1.5.2 Implication elimination, '→E'

$$\frac{A \to B \qquad A}{B} \to\text{E}$$

Implication elimination, →E

If we have `A → B` and `A`, we apply the first proof to the second and obtain `B`. This is modus ponens.

```
theorem lecture03_imp_elim
    (Rains TakeUmbrella : Prop)
    (hRainsUmbrella : Rains → TakeUmbrella)
    (hRains : Rains) :
    TakeUmbrella := by
  have hTakeUmbrella := hRainsUmbrella hRains
  exact hTakeUmbrella
```

`hRainsUmbrella` behaves as a function: it receives a term of type `Rains` and returns a term of type `TakeUmbrella`.

1.6 Conjunction

1.6.1 Conjunction introduction, '∧I'

$$\frac{A \qquad B}{A \wedge B} \wedge\text{I}$$

Conjunction introduction, ∧I

To prove `A ∧ B`, we must construct both components. Applying `And.intro` splits the goal into two cases: first `A`, then `B`.

```
theorem lecture03_and_intro
    (Rains IsCold : Prop)
```

```
    (hRains : Rains)
    (hIsCold : IsCold) :
    Rains ∧ IsCold := by
  apply And.intro
  · exact hRains
  · exact hIsCold
```

The bullets belong to distinct goals, corresponding to the two conjuncts.

1.6.2 Conjunction elimination, '∧Eₗ' and '∧Eᵣ'

$$\frac{A \wedge B}{A}\ \wedge\mathrm{E_l}$$

Left conjunction elimination, ∧Eₗ

$$\frac{A \wedge B}{B}\ \wedge\mathrm{E_r}$$

Right conjunction elimination, ∧Eᵣ

A proof of `A ∧ B` contains a proof of `A` and a proof of `B`. We select the left component with `.left` and the right component with `.right`.

```
theorem lecture03_and_elim_left
    (Rains IsCold : Prop) (hBoth : Rains ∧ IsCold) : Rains := by
  have hRains := hBoth.left
  exact hRains
```

```
theorem lecture03_and_elim_right
    (Rains IsCold : Prop) (hBoth : Rains ∧ IsCold) : IsCold := by
  have hIsCold := hBoth.right
  exact hIsCold
```

Introduction constructs a conjunction from its components; elimination travels in the opposite direction and retrieves a component.

1.7 Disjunction

1.7.1 Disjunction introduction, ‘∨Iₗ‘ and ‘∨Iᵣ‘

$$\frac{A}{A \vee B}\ \vee \mathrm{I_l}$$

Left disjunction introduction, ∨Iₗ

$$\frac{B}{A \vee B}\ \vee \mathrm{I_r}$$

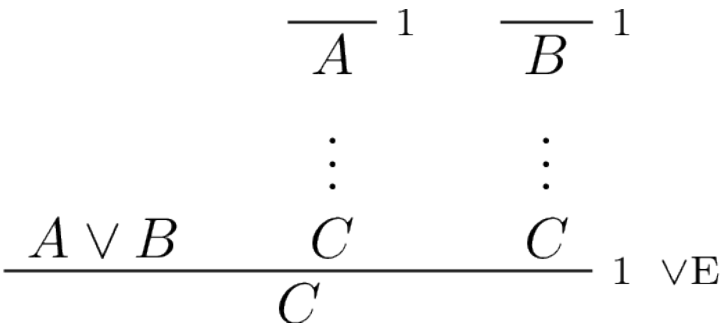
Right disjunction introduction, ∨Iᵣ

To prove `A ∨ B`, it is enough to construct one side, but we must specify which one. `Or.inl` chooses the left side; `Or.inr` chooses the right side.

```
theorem lecture03_or_intro_left
    (Rains Snows : Prop) (hRains : Rains) : Rains ∨ Snows := by
  apply Or.inl
  exact hRains
```

```
theorem lecture03_or_intro_right
    (Rains Snows : Prop) (hSnows : Snows) : Rains ∨ Snows := by
  apply Or.inr
  exact hSnows
```

1.7.2 Disjunction elimination, ‘∨E‘



Disjunction elimination, ∨E

If we have `A ∨ B`, we do not know which side was proved. To obtain one conclusion `C`, we must derive it both in the `A` case and in the `B` case.

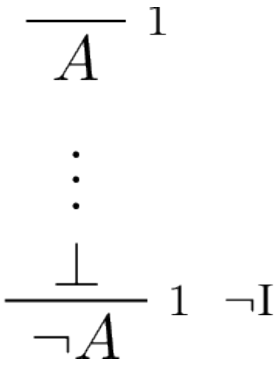
```
theorem lecture03_or_elim
    (Rains Snows TakeUmbrella : Prop)
    (hRainsOrSnows : Rains ∨ Snows)
    (hRainsUmbrella : Rains → TakeUmbrella)
    (hSnowsUmbrella : Snows → TakeUmbrella) :
    TakeUmbrella := by
  cases hRainsOrSnows with
  | inl hRains =>
      have hTakeUmbrella := hRainsUmbrella hRains
      exact hTakeUmbrella
  | inr hSnows =>
      have hTakeUmbrella := hSnowsUmbrella hSnows
      exact hTakeUmbrella
```

`cases` eliminates the disjunction by making all possible forms explicit.

1.8 Negation, truth, and falsity

1.8.1 Negation

In Lean, `¬A` is defined as `A → False`. A proof of `¬A` is therefore a function taking every hypothetical proof of `A` to a contradiction.



Negation introduction, ¬I

To introduce `¬A`, we introduce `A` and construct `False`.

```
theorem lecture03_not_intro
    (DrinkCoffee StayAwake : Prop)
    (hCoffeeAwake : DrinkCoffee → StayAwake)
    (hNotAwake : ¬StayAwake) : ¬DrinkCoffee := by
  intro hDrinkCoffee
  have hStayAwake := hCoffeeAwake hDrinkCoffee
  have hContradiction : False := hNotAwake hStayAwake
  exact hContradiction
```

$$\frac{\neg A \qquad A}{\bot}\ \neg\mathrm{E}$$

Negation elimination, ¬E

To eliminate a negation, we apply `¬A` to `A` and obtain `False`.

```
theorem lecture03_not_elim
    (LabOpen : Prop) (hOpen : LabOpen) (hNotOpen : ¬LabOpen) : False := by
  have hContradiction := hNotOpen hOpen
  exact hContradiction
```

1.8.2 Truth and falsity

$$\frac{}{\top}\ \top\mathrm{I}$$

Truth introduction, ⊤I

`True` has a canonical constructor and requires no assumptions.

```
theorem lecture03_true_intro : True := by
  exact True.intro
```

`exact True.intro` can also be replaced by `trivial`.

$$\frac{\bot}{A}\ \bot\mathrm{E}$$

Falsity elimination, ⊥E

`False` has no constructors. If the context already contains a proof of `False`, we can eliminate falsity and obtain any proposition. This is the principle of *ex falso quodlibet*.

```
theorem lecture03_false_elim
    (Rains : Prop) (hContradiction : False) : Rains := by
  apply False.elim
  exact hContradiction
```

The rule does not let us prove anything arbitrarily: it applies only after a contradiction has been constructed.

1.9 Biconditional

A proof of `A ↔ B` contains two functions: one from `A` to `B` and one from `B` to `A`.

1.9.1 Biconditional introduction, '↔I'

$$\dfrac{\dfrac{\dfrac{}{A}\,1}{\vdots}\quad\dfrac{\dfrac{}{B}\,1}{\vdots}}{\dfrac{B \qquad A}{A \leftrightarrow B}\,1\ \leftrightarrow}$$

Biconditional introduction, ↔I

To prove that "it rains and it is cold" is equivalent to "it is cold and it rains," we construct both directions.

```
theorem lecture03_iff_intro
    (Rains IsCold : Prop) :
    Rains ∧ IsCold ↔ IsCold ∧ Rains := by
  apply Iff.intro
  · intro hRainsAndIsCold
    apply And.intro
    · exact hRainsAndIsCold.right
    · exact hRainsAndIsCold.left
  · intro hIsColdAndRains
    apply And.intro
    · exact hIsColdAndRains.right
    · exact hIsColdAndRains.left
```

1.9.2 Biconditional elimination, '↔Eₗ' and '↔Eᵣ'

$$\frac{A \leftrightarrow B \qquad A}{B} \leftrightarrow \mathrm{E}_l$$

Left-to-right biconditional elimination, ↔Eₗ

$$\frac{A \leftrightarrow B \qquad B}{A} \leftrightarrow \mathrm{E}_r$$

Right-to-left biconditional elimination, ↔Eᵣ

`Iff.mp` extracts the left-to-right direction; `Iff.mpr` extracts the right-to-left direction.

```
theorem lecture03_iff_elim_left
    (EvenNumber DivisibleByTwo : Prop)
    (hEquivalence : EvenNumber ↔ DivisibleByTwo)
    (hEvenNumber : EvenNumber) : DivisibleByTwo := by
  have hDirection := Iff.mp hEquivalence
  have hDivisibleByTwo := hDirection hEvenNumber
  exact hDivisibleByTwo
```

```
theorem lecture03_iff_elim_right
    (EvenNumber DivisibleByTwo : Prop)
    (hEquivalence : EvenNumber ↔ DivisibleByTwo)
    (hDivisibleByTwo : DivisibleByTwo) : EvenNumber := by
  have hDirection := Iff.mpr hEquivalence
  have hEvenNumber := hDirection hDivisibleByTwo
  exact hEvenNumber
```

After extracting the needed direction, we apply it as an ordinary implication.

1.10 Choosing a rule

When reading a proof state, ask a few questions. First: what is the main connective of the goal? It often suggests an introduction rule. Second: which compound proofs are already in the context? They suggest elimination rules.

Lean tactic or term	Natural-deduction reading
`intro h`	introduce an assumption
`apply C`	apply a constructor or rule
`have h := ...`	obtain an intermediate fact
`exact h`	use exactly the required term
`cases h with`	consider the possible cases
`h.left`, `h.right`	eliminate a conjunction
`hAB hA`	eliminate an implication

This vocabulary describes the structure of the term Lean is constructing and makes the relationship between formal rule, natural-language argument, and proof script explicit.

1.11 Working strategy

When constructing a proof:

1. read the goal and assumptions; 2. identify the goal's main connective; 3. choose an introduction rule when the goal suggests one; 4. look for a useful elimination rule in the context; 5. name important intermediate results with `have`; 6. close the goal with `exact` when the required term is available; 7. verify that every case and subgoal has been solved.

`Classroom.lean` presents each rule separately. `Exercises.lean` begins with direct applications and then combines rules into complete arguments. `Solutions.lean` keeps the proof structure explicit so that each Lean step can be connected to its natural-deduction rule.

1.12 Sources and further reading

- Jeremy Avigad, Leonardo de Moura, Soonho Kong, and Sebastian Ullrich, *Theorem Proving in Lean 4*, "Propositions and Proofs": https://lean-lang.org/theorem_proving_in_lean4/Propositions-and-Proofs/.
- Jeremy Avigad, Robert Y. Lewis, and Floris van Doorn, *Logic and Proof*, "Natural Deduction Rules": https://leanprover.github.io/logic_and_proof_lean3/nd_quickref.html.
- Philip Wadler, "Propositions as Types," *Communications of the ACM* 58(12), 2015: https://homepages.inf.ed.ac.uk/wadler/papers/propositions-as-types/propositions-as-types.pdf.
- NASA Science, "Io: Facts": https://science.nasa.gov/jupiter/jupiter-moons/io/facts/.