P02_Logic

Logic

Logic — Contents

Propositional logic: notation

We write P, Q, R, ... for propositions (statements that are true or false).

Propositional logic: notation

We write P, Q, R, ... for propositions (statements that are true or false).

Symbol	Name	Read as
$P \to Q$	implication	"if P then Q"
$P \wedge Q$	conjunction	"P and Q"
$P \vee Q$	disjunction	"P or Q"
$\neg P$	negation	"not P"
$P \leftrightarrow Q$	biconditional	"P if and only if Q"
$\top$ / $\bot$	truth / falsity	"true" / "false"

Propositional logic: notation

We write P, Q, R, … for propositions (statements that are true or false).

Symbol	Name	Read as
$P \to Q$	implication	"if P then Q"
$P \wedge Q$	conjunction	"P and Q"
$P \vee Q$	disjunction	"P or Q"
$\neg P$	negation	"not P"
$P \leftrightarrow Q$	biconditional	"P if and only if Q"
$\top$ / $\bot$	truth / falsity	"true" / "false"

$P \to Q$ only asserts: *if* P holds, *then* Q follows. In classical logic, $P \to Q \equiv \neg P \vee Q$, so it is true whenever P is false ("vacuously").

Let's look at this in Lean.

Fundamentals: takeaways

1. Lean has two proof styles: **tactic mode** (`by` block) and **term mode**. Most Mathlib proofs use tactic mode for complex arguments and term mode for short ones.
2. A **tactic proof** transforms the goal step by step until nothing remains. The goal state shows what you still need to prove; the context shows what you have.
3. A **term proof** chains function applications directly, without intermediate goals. Where tactic mode says "rewrite, then simplify," term mode hands Lean the finished expression.

Fundamentals: tactic reference

Tactic	Effect
`rfl`	Close a goal of the form `a = a`.
`assumption`	Close the goal with a hypothesis from the context.
`exact e`	Close the goal with the term `e`.
`exact?`	Ask Lean to search for a closing term.
`intro h`	Prove `P → Q` by assuming `P` as `h`.
`revert h`	Move `h` back into the goal (inverse of `intro`).
`rw [h]`	Rewrite the goal using `h`; use `← h` for reverse.
`rw [h] at k`	Rewrite hypothesis `k` instead of the goal.
`nth_rw n [h]`	Rewrite only the *n*-th occurrence.

Logic — Contents

Forward and backward reasoning

Consider a chain $A \to B \to C \to D$. You know A and want to prove D.

Forward and backward reasoning

Consider a chain $A \to B \to C \to D$. You know A and want to prove D.

1. **Forward reasoning** Start from what you know.

   I know A.
   Since $A \to B$, I have B.
   Since $B \to C$, I have C.
   Since $C \to D$, I have D.

Forward and backward reasoning

Consider a chain $A \to B \to C \to D$. You know A and want to prove D.

1. **Forward reasoning** Start from what you know.

   I know A.
   Since $A \to B$, I have B.
   Since $B \to C$, I have C.
   Since $C \to D$, I have D.

2. **Backward reasoning** Start from the goal.

   I want D.
   It suffices to show C.
   It suffices to show B.
   It suffices to show A. Done.

Forward and backward reasoning

Consider a chain $A \to B \to C \to D$. You know A and want to prove D.

1. **Forward reasoning** Start from what you know.

   I know A.
   Since $A \to B$, I have B.
   Since $B \to C$, I have C.
   Since $C \to D$, I have D.

2. **Backward reasoning** Start from the goal.

   I want D.
   It suffices to show C.
   It suffices to show B.
   It suffices to show A. Done.

Both strategies produce the same logical content. Forward reasoning reads like a narrative; backward reasoning reads like a plan.

Let's look at this in Lean.

Logic — Contents

Logical connectives: proving and using

Connectives combine propositions into compound statements: $P \wedge Q$, $P \vee Q$, $P \leftrightarrow Q$. For each, there are two questions:

1. How do you **prove** a statement involving the connective?
2. How do you **use** a hypothesis involving the connective?

Logical connectives: proving and using

Connectives combine propositions into compound statements: $P \wedge Q$, $P \vee Q$, $P \leftrightarrow Q$. For each, there are two questions:

1. How do you **prove** a statement involving the connective?
2. How do you **use** a hypothesis involving the connective?

	To prove	**To use as hypothesis**
$P \wedge Q$	Prove P and prove Q separately.	Extract P, extract Q, or both.
$P \vee Q$	Prove P or prove Q (choose one side).	Consider both cases: assume P, then assume Q.
$P \leftrightarrow Q$	Prove $P \to Q$ and prove $Q \to P$.	Use either direction as an implication; or substitute P for Q.

Let's look at this in Lean.

Connectives: takeaways

1. Every connective has two faces: an *introduction* rule (how to prove it) and an *elimination* rule (how to use it). The right tactic depends only on which side you are on.
2. This build-vs-disassemble pattern is not specific to $\wedge/\vee/\leftrightarrow$. It recurs for $\exists$, for inductive types, and for structures—learning it once pays off throughout the course.

Connectives: tactic reference

Tactic	Applies to	Result
`constructor`, $\langle \cdot, \cdot \rangle$	Goal: $P \wedge Q$ or $P \leftrightarrow Q$	Two subgoals
`left` / `right`	Goal: $P \vee Q$	Goal becomes P (or Q)
`obtain` / `rcases`	Hyp: $P \wedge Q$, $P \vee Q$, $\exists$	Components in context
`.1` / `.2`	Hyp: $P \wedge Q$	Left or right part
`.mp` / `.mpr`	Hyp: $P \leftrightarrow Q$	Forward or backward implication
`trans`	Goal: $a = c$ or $a \leftrightarrow c$	Two goals via intermediate
`tfae_have` / `tfae_finish`	Goal: `TFAE` list	Prove selected implications

Logic — Contents

Negation and contradiction

Negation

$\neg P$ means: P does not hold. A standard way to prove $\neg P$ is to assume P and derive a contradiction ($\bot$).

Negation and contradiction

Negation

$\neg P$ means: P does not hold. A standard way to prove $\neg P$ is to assume P and derive a contradiction ($\bot$).

1. **Ex falso quodlibet** If you have established $\bot$, you may conclude any proposition Q.

Negation and contradiction

Negation

$\neg P$ means: P does not hold. A standard way to prove $\neg P$ is to assume P and derive a contradiction ($\bot$).

1. **Ex falso quodlibet** If you have established $\bot$, you may conclude any proposition Q.

2. **Proof by contradiction**
   - To prove P: assume $\neg P$ and derive $\bot$. Conclude P.
   - This is **not** the same as proving $\neg P$. Proving $\neg P$ means: assume P, derive $\bot$, conclude $\neg P$. That step is constructively valid.
   - Proof by contradiction (for a positive P) requires $\neg\neg P \to P$.

Classical vs. intuitionistic logic

In **intuitionistic** (constructive) logic, a proof of $P \vee Q$ must exhibit which disjunct holds. In **classical** logic, $P \vee \neg P$ holds for every P without constructing a witness.

Classical vs. intuitionistic logic

In **intuitionistic** (constructive) logic, a proof of $P \vee Q$ must exhibit which disjunct holds. In **classical** logic, $P \vee \neg P$ holds for every P without constructing a witness.

Classical axioms

Equivalent over intuitionistic logic: **excluded middle** ($P \vee \neg P$) and **double negation elimination** ($\neg\neg P \to P$). Adopting either one yields classical logic.

Classical vs. intuitionistic logic

In **intuitionistic** (constructive) logic, a proof of $P \vee Q$ must exhibit which disjunct holds. In **classical** logic, $P \vee \neg P$ holds for every P without constructing a witness.

Classical axioms

Equivalent over intuitionistic logic: **excluded middle** ($P \vee \neg P$) and **double negation elimination** ($\neg\neg P \to P$). Adopting either one yields classical logic.

1. **Constructively valid** $P \to \neg\neg P$ (double negation introduction); $(P \to Q) \to (\neg Q \to \neg P)$ (contrapositive).

Classical vs. intuitionistic logic

In **intuitionistic** (constructive) logic, a proof of $P \vee Q$ must exhibit which disjunct holds. In **classical** logic, $P \vee \neg P$ holds for every P without constructing a witness.

Classical axioms

Equivalent over intuitionistic logic: **excluded middle** ($P \vee \neg P$) and **double negation elimination** ($\neg\neg P \to P$). Adopting either one yields classical logic.

1. **Constructively valid** $P \to \neg\neg P$ (double negation introduction); $(P \to Q) \to (\neg Q \to \neg P)$ (contrapositive).

2. **Requires classical logic** $\neg\neg P \to P$ (double negation elimination); $(\neg Q \to \neg P) \to (P \to Q)$ (reverse contrapositive); proof by contradiction for positive statements.

Let's look at this in Lean.

Negation: takeaways

1. Since $\neg P$ is $P \to \bot$, the implication tactics (`intro`, `apply`, `exact`) handle negation directly. No special machinery is needed for the constructive case.

Habit. Try the constructive approach first: unfold $\neg P$ as $P \to \bot$ and work with `intro`/`apply`. Reach for `by_contra` or `by_cases` only when you genuinely need to assume the negation of what you are proving.

Negation: tactic reference

Tactic	Effect
`contradiction`	Close the goal when the context contains conflicting hypotheses (e.g. $h : P$ and $h' : \neg P$).
`exfalso`	Change the current goal to `False`, letting you derive a contradiction manually.
`by_contra h`	Assume $\neg\, goal$ as `h` and prove `False` (classical).
`by_cases h : P`	Split into two branches: one where P holds, one where $\neg P$ holds (classical).
`push_neg`	Push negations inward (uses classical logic): e.g. $\neg(P \wedge Q) \rightsquigarrow P \to \neg Q$, $\neg(P \to Q) \rightsquigarrow P \wedge \neg Q$, $\neg\neg P \rightsquigarrow P$.
`trivial`	Try simple closers (`rfl`, `contradiction`, `assumption`, and others).

Logic — Contents

Quantifiers and witnesses

1 Universal quantification

- $\forall x,\ P(x)$ means: for every x, the statement $P(x)$ holds.
- To prove $\forall x,\ P(x)$: let x be arbitrary and prove $P(x)$.
- To use $\forall x,\ P(x)$: instantiate it at a specific value to obtain $P(a)$.

Quantifiers and witnesses

1 Universal quantification

- $\forall x,\ P(x)$ means: for every x, the statement $P(x)$ holds.
- To prove $\forall x,\ P(x)$: let x be arbitrary and prove $P(x)$.
- To use $\forall x,\ P(x)$: instantiate it at a specific value to obtain $P(a)$.

2 Existential quantification

- $\exists x,\ P(x)$ means: there exists at least one x such that $P(x)$ holds.
- To prove $\exists x,\ P(x)$: exhibit a concrete a and prove $P(a)$.
- To use $\exists x,\ P(x)$: introduce a witness a with $P(a)$ and continue.

Quantifiers and witnesses

1 Universal quantification

- $\forall x,\ P(x)$ means: for every x, the statement $P(x)$ holds.
- To prove $\forall x,\ P(x)$: let x be arbitrary and prove $P(x)$.
- To use $\forall x,\ P(x)$: instantiate it at a specific value to obtain $P(a)$.

2 Existential quantification

- $\exists x,\ P(x)$ means: there exists at least one x such that $P(x)$ holds.
- To prove $\exists x,\ P(x)$: exhibit a concrete a and prove $P(a)$.
- To use $\exists x,\ P(x)$: introduce a witness a with $P(a)$ and continue.

3 Extensionality

- Two functions $f, g : X \to Y$ are equal iff $f(x) = g(x)$ for all x.
- Two sets $A, B \subseteq X$ are equal iff $x \in A \Leftrightarrow x \in B$ for all x.

Let's look at this in Lean.

Quantifiers: takeaways

1. In Lean, $\forall$ and $\to$ are the same type former. All three spellings `∀ x, P x`, `(x : α) → P x`, and `(x : α) : P x` (in signatures) mean the same thing. So `intro`/`apply` work for $\forall$ exactly as for $\to$.
2. An existential $\exists\, x,\ P(x)$ is a pair $\langle x, h\rangle$. So `use`/`obtain` work for $\exists$ exactly as `constructor`/`obtain` work for $\wedge$.

Habit. When you encounter an unfamiliar logical symbol, ask what its type is. If it is a function type, `intro` and `apply` work. If it is built with $\langle\cdot,\cdot\rangle$, `constructor` and `obtain` work.

Outlook: automation (optional)

Optional outlook

This is perspective material, not part of the core manual-proof workflow.

- `tauto` closes propositional tautologies in one line.
- `grind` (Lean 4.22+) goes further and can also handle quantifiers and arithmetic.

Understanding the manual proofs tells you *why*; automation handles the *that*.

Quantifiers: tactic reference

Tactic	Applies to	Result
`use a`	Goal: $\exists\, x,\ P(x)$	Goal becomes $P(a)$
`obtain ⟨w, hw⟩ := h`	Hyp: $h : \exists\, x,\ P(x)$	Adds w and $hw : P(w)$
`choose f hf using h`	Hyp: $h : \forall x, \exists y,\ P\, x\, y$	Adds f and hf
`ext x`	Goal: $f = g$	Goal becomes $f\, x = g\, x$