Lean 4 Summary Sheet

1. Term Builders for `Prop`

These are constructors used to explicitly build terms inhabiting Prop in Lean4. The number of underscores following the constructor indicates the number of terms that can be passed to it. Note it can be useful, when using tactics, to only partially apply some constructors.

1. `And.intro _ _`

   e.g. Given terms $p : P$ and $q : Q$ the term And.intro $p\ q : P \wedge Q$.

   This can also be written $\langle p, q \rangle : P \wedge Q$

2. `And.left _`

   e.g. Given a term $t : P \wedge Q$, the term And.left $t : P$

   This can also be written t.left $: P$

3. `And.right _`

   e.g. Given a term $t : P \wedge Q$, the term And.right $t : Q$

   This can also be written t.right $: Q$

4. `Or.intro_left _ _`

   e.g. Given a term $p : P$ and a Prop Q the term Or.intro_left $Q\ p : P \vee Q$

5. `Or.intro_right _ _`

   e.g. Given a term $p : P$ and a Prop Q the term Or.intro_right $Q\ p : Q \vee P$

6. `Or.elim _ _ _`

   e.g. Given terms $t : P \vee Q$, $f : P \to R$, and $g : Q \to R$ the term Or.elim $t\ f\ g : R$

7. `fun _ => _`

   e.g. Given an identifier (any lower case english string) say $t : P$ and a term $e : Q$ we can construct fun $t : P => e$ which has type $P \to Q$

   This can be written without the explicit typing on the variable t. It can also be written with a λ instead of fun

   $\lambda t => e$ is a term of type $P \to Q$. Lean can infer the type of t.

   Notice the use of $=>$ instead of the . used in our on-paper lambda calculus.

2. Unicode Symbols in VSCode

Type a backslash \ followed by the keyword to insert the symbol:

Name	Lean Input	Symbol	Meaning
And	`\and`	$\wedge$	Logical AND
Or	`\or`	$\vee$	Logical OR
Not	`\not`	$\neg$	Logical NOT
Implies	`\to`	$\to$	Implication
Forall	`\forall`	$\forall$	Universal quantifier
Exists	`\exists`	$\exists$	Existential quantifier
Natural numbers	`\bn`	$\mathbb{N}$	Set of naturals
Alpha	`\alpha`	α	Greek letter alpha
Beta	`\beta`	β	Greek letter beta
Turnstile	`\vdash`	$\vdash$	Provability symbol
Lambda	`\lambda`	λ	Lambda abstraction
Left Angle bracket	`\langle`	$\langle$	AND introduction
Right Angle bracket	`\rangle`	$\rangle$	AND introduction

3. Keywords

Avoid using these as identifiers (names) for terms, types, or variables.

- `variable`

  This is used to declare variables inhabiting particular types.

- `theorem`

  This is used to state a theorem in Lean. It has the following pattern:

  theorem <name> <list of hypotheses> : <goal-type> := <proof-term>

- `example`

  This is used to state an anonymous theorem in Lean. It has the following pattern:

  example <list of hypotheses> : <goal-type> := <proof-term>

- `def`
- `with`
- `match`

4. Tactics for Proofs in `Prop`

Rather than writing proof-terms explicitly, these tactics help authoring proofs about propositions.

- `intro _ _ _ ...`

  This mimics the use of the deduction theorem. If the goal is an implication, then it uses the name(s) passed to intro to put a term of the antecedent type in the current context. If there are nested implications, then intro can be passed a number of identifiers to move all the antecedents to the current context in one line.

  e.g. If the current goal is $A \to (B \to (C \wedge D \to E))$, then we can apply intro a b t which will update the goal to E and put three terms $a : A$, $b : B$, and $t : C \wedge D$ into the context.

  It takes care of all the implication introductions behind the scenes.

- `apply _`

- `exact _`

  e.g. If the goal of the current proof-state is P and $t : P$, then exact t completes the proof.

- `have _ := _`

  e.g. This tactic introduces an intermediate term into the context. It gives a name to a subproof used on the way to the goal. It doesn't change the goal of the proof-state. If $e : P$, then have $t := e$ introduces the term t of type P into the context.

  If the proof is particularly long, then using have can break the proof up into manageable steps.