Predicate Logic in Lean

Heshan Du
University of Nottingham Ningbo China

October 2024

Reading

- Thorsten Altenkirch, *Introduction to Formal Reasoning*, 2023.
    - Chapter 4. Predicate Logic

Predicates, Relations, and Quantifiers

- Predicate logic extends propositional logic. We can use it to talk about objects and their properties.
- The objects are organised in *types*, such as,
  - $\mathbb{N}$: *Type*, the type of natural numbers $\{0, 1, 2, 3, \ldots\}$
  - *bool* : *Type*, the type of booleans $\{tt, ff\}$
  - lists over a given A : *Type*, *list* A : *Type*.
- To avoid talking about specific types, we introduce some type variables:

```
variables A B C : Type
```

Predicates

- A predicate is just another word for a property.
  - E.g., we may use $\mathit{Prime} : \mathbb{N} \to \mathit{Prop}$ to express that a number is a prime number.
  - We can form propositions such as *Prime 3* and *Prime 4*, the first one should be provable while the negation of the second holds.
- Predicates may have several inputs in which case we usually call them relations.
  - $\leq : \mathbb{N} \to \mathbb{N} \to \mathit{Prop}$, e.g., $2 \leq 3$
  - $\mathit{inList} : A \to \mathit{list}\ A \to \mathit{Prop}$, e.g., $\mathit{inList}\ 1\ [1, 2, 3]$
- In the sequel, we will use some generic predicates, e.g.,

```
variables PP QQ : A -> Prop
```

Quantifiers 1

The most important innovation of predicate logic are the quantifiers, which we can use to form new propositions:

- universal quantification ($\forall$), read $\forall\, x : A,\ PP\ x$ as all x in A satisfy $PP\ x$.
- existential quantification ($\exists$), read $\exists\, x : A,\ PP\ x$ as there is an x in A satisfying $PP\ x$.

Quantifiers 2

- Both universal quantifier and existential quantifier bind weaker than any other propositional operator.
- We read $\forall\, x\ : A,\ PP\ x \land Q$ as $\forall\, x\ : A,\ (PP\ x \land Q)$.
- We use parentheses to limit the scope, e.g., $(\forall x\ : A,\ PP\ x) \land Q$

Bound Variables

- It is important to understand bound variables, essentially they work like scoped variables in programming.
- We can shadow variables as in $\forall\, x\ : A,\ (\exists x : A,\ PP\ x) \wedge QQ\ x$.
- Bound variables can consistently renamed, e.g., $\forall\, y\ : A,\ (\exists z : A,\ PP\ z) \wedge QQ\ y$.
- *Shadowing variables should be avoided because it confuses the human reader*.

A Primitive Proposition

- We have a new primitive proposition equality (=).
- Given $a\ b : A$, we write $a = b$, read as a is equal to b.

The Universal Quantifier

- To prove that a proposition of the form $\forall x : A,\ PP\ x$ holds, we assume that there is given an arbitrary element a in A and prove it for this generic element, i.e., to prove $PP\ a$, we use *assume a* to do this.
- If we have an assumption $h : \forall x : A,\ PP\ x$ and our current goal is $PP\ a$ for some $a : A$, then we can *apply h* to prove our goal.
- Usual we have some combination of implication and for all, like $h : \forall x : A,\ PP\ x \to QQ\ x$ and now if our current goal is $QQ\ a$ and we invoke *apply h*, Lean will instantiate x with a and it remains to show $PP\ a$.

Example 1

How to prove
$(\forall x\ : A,\ PP\ x) \to (\forall y\ : A,\ PP\ y \to QQ\ y) \to \forall z\ : A,\ QQ\ z$ in Lean?

```
variable A : Type
variables PP QQ : A → Prop

example : (∀ x: A, PP x)
    → (∀ y: A, PP y → QQ y)
    → ∀ z : A, QQ z :=
begin

end
```

Example 2

Let us prove a logical equivalence involving $\forall$ and $\wedge$:

$$(\forall x \ : A, PP\ x \wedge QQ\ x) \leftrightarrow (\forall x \ : A, PP\ x) \wedge (\forall x \ : A, QQ\ x)$$

```
variable A : Type
variables PP QQ : A → Prop

example : (∀ x: A, PP x ∧ QQ x)
        ↔ (∀ x: A, PP x) ∧ (∀ x: A, QQ x) :=
begin

end
```

The Existential Quantifier

- To prove a proposition of the form $\exists x : A, PP\ x$, it is enough to prove $PP\ a$ for any $a : A$.
- We use *existsi a* for this, and we are left to prove $PP\ a$.
- Note that a can be any expression of type A not necessarily a variable.
- To use an assumption of the form $h : \exists x : A, P\ x$, we use *cases h with x px*, which replaces h with two assumptions $x : A$ and $px : P\ x$.

Example 3

How to prove
$(\exists x\ : A, PP\ x) \to (\forall y\ : A, PP\ y \to QQ\ y) \to \exists z\ : A, QQ\ z$?

```
variable A : Type
variables PP QQ : A → Prop

example : (∃ x : A, PP x)
          → (∀ y: A, PP y → QQ y)
          → (∃ z: A, QQ z) :=
begin

end
```

Example 4

How to prove
$(\exists x : A, PP\ x \vee QQ\ x) \leftrightarrow (\exists x : A, PP\ x) \vee (\exists x : A, QQ\ x)$?

```
variable A : Type
variables PP QQ : A → Prop

example : (∃ x: A, PP x ∨ QQ x)
        ↔ (∃ x: A, PP x) ∨ (∃ x: A, QQ x) :=
begin

end
```

Another Currying Equivalence 1

- You may have noticed that the way we prove propositions involving $\to$ and $\forall$ is very similar.
- In both cases, we use *assume* to introduce an assumption, and then use *apply* to prove the current goal.
- The way $\land$ and $\exists$ behave is similar.
- In both cases, we prove them using *constructor* where we construct two components, then use *cases* with two components, which replaces the assumption by its two components.

Another Currying Equivalence 2

- The similarity can be seen by establishing another currying-style equivalence.
- While currying in propositional logic had the form

$$P \wedge Q \to R \leftrightarrow P \to Q \to R$$

where we turn a conjunction into an implication, currying for predicate logic has the form

$$(\exists x \ : A, QQ\ x) \to R \leftrightarrow (\forall x \ : A, QQ\ x \to R)$$

Example 5

How to prove $(\exists x : A, PP\ x) \to R \leftrightarrow (\forall x : A, PP\ x \to R)$?

```
variable A : Type
variable R : Prop
variables PP QQ : A → Prop

theorem curry_pred: (∃ x: A, PP x) → R
                    ↔ (∀ x: A, PP x → R) :=
begin

end
```

Equality

- There is a generic relation which can be applied to any type: *equality*.
- Given $a\ b\ : A$, we can construct $a = b\ : Prop$ expressing that a and b are equal.
- We can prove that everything is equal to itself using the tactic *reflexivity*.

Example 6

How to prove $\forall\ x\ : A,\ x = x$ in Lean?

```
variable A : Type

example : ∀ x : A, x = x :=
begin
    
end
```

Example 7

- If we have assumed an equality $h : a = b$, we can use it to *rewrite* a into b in the goal.
- This is, if our goal is $PP\ a$, we say *rewrite* h, and this changes the goal into $PP\ b$.
- How to prove $\forall\ x\ y\ : A,\ x = y \to PP\ y \to PP\ x$?

```
variable A : Type
variables PP QQ : A → Prop

example : ∀ x y : A, x = y → PP y → PP x :=
begin

end
```

Example 8

- Sometimes we want to use the equality in the other direction, this is, we want to replace *b* by *a*.
- In this case, we use *rewrite* $\leftarrow$ *h*.
- How to prove $\forall x\ y\ : A,\ x = y \rightarrow PP\ x \rightarrow PP\ y$?

```
variable A : Type
variables PP QQ : A → Prop

example : ∀ x y : A, x = y → PP x → PP y :=
begin
    
end
```

Properties of Equality

Equality is an *equivalence relation*, which means that it is

- reflexive ($\forall\ x\ : A,\ x = x$),
- symmetric ($\forall\ x\ y\ : A,\ x = y \to y = x$),
- transitive ($\forall\ x\ y\ z\ : A,\ x = y \to y = z \to x = z$).

Proving the Properties

- We have already shown *reflexivity* using the appropriately named tactic.
- We can show *symmetry* and *transitivity* using *rewrite*.

Proving Symmetry

How to prove $\forall\ x\ y\ : A,\ x = y \to\ y = x$ in Lean?

```
variable A : Type

theorem sym_eq : ∀ x y : A, x = y → y = x :=
begin

end
```

Proving Transitivity

How to prove $\forall\ x\ y\ z\ : A,\ x = y \to y = z \to x = z$ in Lean?

```
variable A : Type

theorem trans_eq: ∀ x y z : A, x = y → y = z → x = z :=
begin
    
end
```

Rewrite an assumption

- Sometimes we want to use an equality not to rewrite the goal but to rewrite another assumption.
- e.g., by saying *rewrite xy at yz*, we are using *xy* to rewrite *yz*.
- The same works for *rewrite* $\leftarrow$.
- Try it in the proof below.

```
variable A : Type

theorem trans_eq: ∀ x y z : A, x = y → y = z → x = z :=
begin

end
```

Tactics: *symmetry* and *transitivity*

- Actually Lean already has built-in tactics to deal with symmetry and transitivity.
- Try them in proving the theorems *sym_eq* and *trans_eq*.

Proving Theorem *trans_eq* using *calc*

- Below we are not going to use *transitivity*, but use a special format for equational proofs indicated by *calc*.
- e.g.,

```
calc
   x = y    :  by exact xy
   ...= z   : by exact yz,
```

- After *calc* we prove a sequence of equalities where each step is using *by* followed by some tactics.
- Any subsequent line starts with ... which stands for the last expression of the previous line, in this case *y*.
- Try *calc* in proving the theorem *trans_eq*.

Congruence

- There is one more property we expect from equality.
- Assume we have a function $f\ :\ A \to B$ and we know that $x = y$ for some $x\ y\ : A$, then we want to be able to conclude that $f\ x = f\ y$.
- We say that equality is a congruence.

Example 9

How to prove $\forall f\ : A \to B,\ \forall x\ y\ : A,\ x = y \to f\ x = f\ y$?

```
variables A B: Type

theorem congr_argf: ∀ f : A → B, ∀ x y : A, x = y → f x = f y :=
begin

end
```

Classical Predicate Logic

- We can use classical logic in predicate logic, even though the explanation using truth tables does not work any more.
- There are predicate logic counterparts of the de Morgan laws, which now say that you can move negation through a quantifier by negating the component and switching the quantifier.
- Again, one of them is provable intuitionistically.

de Morgan Law 1

How to prove $\neg(\exists x : A, PP\ x) \leftrightarrow \forall\ x : A, \neg PP\ x$?

```
variables A B: Type
variables PP QQ : A → Prop

theorem dm1_pred : ¬ (∃ x : A, PP x) ↔ ∀ x : A, ¬ PP x :=
begin

end
```

de Morgan Law 2

How to prove $\neg(\forall x : A,\ PP\ x) \leftrightarrow \exists x : A,\ \neg PP\ x$?

```
variables A B: Type
variables PP QQ : A → Prop

theorem dm2_pred : ¬ (∀ x : A, PP x) ↔ ∃ x : A, ¬ PP x :=
begin

end
```

Summary of Tactics

Here is the summary of basic tactics for predicate logic:

	How to prove?	How to use?
$\forall$	*assume h*	*apply h*
$\exists$	*existsi a*	*cases h with x p*
$=$	*reflexivity*	*rewrite h rewrite* $\leftarrow$ *h*

Note that the *a* after *existsi* can be any expression of type *A*, while *h x p* are variables.