Section 4 Solutions – Theorem Proving in Lean 4

Chun Ding • 20 May 2025

```
variable (α : Type) (p q : α → Prop)

example : (∀ x, p x ∧ q x) ↔ (∀ x, p x) ∧ (∀ x, q
        x) :=
  Iff.intro
    (fun h => ⟨fun x => (h x).left, fun x => (h
        x).right⟩)
    (fun ⟨hp, hq⟩ x => ⟨hp x, hq x⟩)
example : (∀ x, p x → q x) → (∀ x, p x) → (∀ x, q
        x) :=
  fun hpq hp x => hpq x (hp x)

example : (∀ x, p x) ∨ (∀ x, q x) → ∀ x, p x ∨ q x :=
  fun h x => Or.elim h (fun hp => Or.inl (hp x)) (fun
        hq => Or.inr (hq x))
```

```
variable (α : Type) (p q : α → Prop)
variable (r : Prop)

example : α → ((∀ x : α, r) ↔ r) :=
  fun x => Iff.intro (fun hr  => hr x) (fun hr _ =>
        hr)

example : (∀ x, p x ∨ r) ↔ (∀ x, p x) ∨ r :=
  Iff.intro
    (fun h => Or.elim (Classical.em r)
      (fun hr => Or.inr hr)
      (fun hnr => Or.inl (fun x => Or.resolve_right (h
        x) hnr)))
    (fun h x => Or.elim h (fun hp => Or.inl (hp x)) (f
        un hr => Or.inr hr))
example : (∀ x, r → p x) ↔ (r → ∀ x, p x) :=
  Iff.intro
    (fun h hr x => h x hr)
    (fun h x hr => h hr x)
```

```
example (h : ∀ x : men, shaves barber x ↔ ¬ shaves x
        x) : False :=
  let h_barber := h barber
  (em (shaves barber barber)).elim
    (λ h_shaves => h_barber.mp h_shaves h_shaves)
    (λ h_not_shaves => h_not_shaves (h_barber.mpr
        h_not_shaves))
```

```
open Classical

variable (α : Type) (p q : α → Prop)
variable (r : Prop)

example : (∃ x : α, r) → r :=
  fun ⟨_, hr⟩ => hr
example (a : α) : r → (∃ x : α, r) :=
  fun hr => ⟨a, hr⟩
example : (∃ x, p x ∧ r) ↔ (∃ x, p x) ∧ r :=
  Iff.intro
    (fun ⟨x, ⟨hp, hr⟩⟩ => ⟨⟨x, hp⟩, hr⟩)
    (fun ⟨⟨x, hp⟩, hr⟩ => ⟨x, ⟨hp, hr⟩⟩)
example : (∃ x, p x ∨ q x) ↔ (∃ x, p x) ∨ (∃ x, q
        x) :=
  Iff.intro
    (fun |⟨x, Or.inl hp⟩ => Or.inl ⟨x, hp⟩ | ⟨x,
        Or.inr hq⟩ => Or.inr ⟨x, hq⟩)
    (fun | Or.inl ⟨x, hp⟩ => ⟨x, Or.inl hp⟩ | Or.inr
        ⟨x, hq⟩ => ⟨x, Or.inr hq⟩)

example : (∀ x, p x) ↔ ¬ (∃ x, ¬ p x) := Iff.intro
  (fun h => fun ⟨x, hnpx⟩ => hnpx (h x))
  (fun h x => Or.elim (Classical.em (p x))
    (fun hp => hp)
    (fun hnp => False.elim (h ⟨x, hnp⟩)))

example : (∃ x, p x) ↔ ¬ (∀ x, ¬ p x) := Iff.intro
  (fun ⟨x, hp⟩ => fun h => (h x hp))
  (fun h : ¬ (∀ x, ¬ p x) => byContradiction fun h1 :
        ¬ (∃ x, p x) =>
    h (fun hx hpx => h1 ⟨hx, hpx⟩))

example : (¬ ∃ x, p x) ↔ (∀ x, ¬ p x) := Iff.intro
  (fun h x hpx => h ⟨x, hpx⟩)
```

```
  (fun h => fun ⟨hx, hp⟩ => h hx hp)
example : (¬ ∀ x, p x) ↔ (∃ x, ¬ p x) := Iff.intro
  (fun h => byContradiction fun h1 : ¬ ∃ x, ¬ p x => h
        (fun x => Or.elim (Classical.em (p x))
    (fun hp => hp)
    (fun hnp => False.elim (h1 ⟨x, hnp⟩))))
  (fun ⟨x, hnp⟩ h => hnp (h x))
example : (∀ x, p x → r) ↔ (∃ x, p x) → r := Iff.intro
  (fun h ⟨x, hp⟩ => h x hp)
  (fun h x hp => h ⟨x, hp⟩)

example (a : α) : (∃ x, p x → r) ↔ (∀ x, p x) → r :=
        Iff.intro
  (fun ⟨x, hp⟩ h => hp (h x))
  (fun h => Or.elim (Classical.em (∀ x, p x))
    (fun y => ⟨a, λ _ =>h y⟩)
    (fun hnp =>
      have lemmap: ∃ x, ¬ p x := byContradiction
        fun h1 : ¬ ∃ x, ¬ p x => hnp (fun x => Or.elim
        (Classical.em (p x))
          (fun hpx => hpx)
          (fun hnpx => False.elim (h1 ⟨x, hnpx⟩)))
      let ⟨x0, hnp0⟩ := lemmap
      ⟨x0, fun hp => False.elim (hnp0 hp)⟩))

example (a : α) : (∃ x, r → p x) ↔ (r → ∃ x, p x) :=
        Iff.intro
  (fun ⟨x, hr⟩ h => ⟨x, hr h⟩)
  (fun h => Or.elim (Classical.em r)
    (fun hr =>
      match h hr with
      | ⟨x, hp⟩ => ⟨x, λ _ ↦ hp⟩)
    (fun hnr => ⟨a, λ hr ↦ absurd hr hnr⟩))
```