Lezione 5: Regole dei quantificatori

Stefano M. Nicoletti

1 Lezione 5: Regole dei quantificatori

Registrazione video della Lezione 5

1.1 Scopo della lezione

Riprendiamo le formalizzazioni della Lezione 4 e introduciamo le regole di deduzione naturale per ∀ e ∃. Prima completiamo gli esempi con funzioni e quantificatori annidati. Poi studiamo come costruire e usare dimostrazioni universali ed esistenziali, comprese le condizioni che rendono valide le quattro regole. Infine combiniamo i quantificatori con i connettivi già noti.

Come ripasso, confrontiamo quattro forme ricorrenti:

```
variable (Cosa : Type)
variable (Studentessa Legge Comprende : Cosa → Prop)

#check ∀ x : Cosa, Studentessa x → Legge x
#check ∃ x : Cosa, Studentessa x ∧ Legge x
#check ∀ x : Cosa, Studentessa x → Legge x → Comprende x
#check ∃ x : Cosa, Studentessa x ∧ Legge x ∧ Comprende x
```

1.2 Funzioni dentro le formule

I termini costruiti mediante funzioni possono comparire dentro predicati, relazioni e quantificatori:

```
variable (Cosa : Type)
variable (ipazia : Cosa)
variable (Studentessa Legge Curiosa : Cosa → Prop)
variable (madreDi : Cosa → Cosa)
variable (Conosce : Cosa → Cosa → Prop)

#check Curiosa (madreDi ipazia)

#check ∀ x : Cosa,
  Studentessa x → Conosce x (madreDi x)

#check ∃ x : Cosa,
  Studentessa x ∧ Legge (madreDi x)
```

Le parentesi rendono visibile l'ordine della costruzione: prima formiamo il termine `madreDi x`, poi lo usiamo come argomento del predicato `Legge` o della relazione `Conosce`.

1.3 Quantificatori annidati

Una formula può contenere più quantificatori. Consideriamo la relazione `Conosce : Cosa → Cosa → Prop`:

```
variable (Cosa : Type)
variable (Persona : Cosa → Prop)
variable (Conosce : Cosa → Cosa → Prop)

-- Ogni persona conosce qualcuno.
#check ∀ x : Cosa, Persona x →
  ∃ y : Cosa, Persona y ∧ Conosce x y

-- Qualcuno conosce ogni persona.
#check ∃ x : Cosa, Persona x ∧
  ∀ y : Cosa, Persona y → Conosce x y

-- Nessuno conosce ogni persona.
#check ∃¬ x : Cosa, Persona x ∧
  ∀ y : Cosa, Persona y → Conosce x y
```

Le prime due proposizioni non sono equivalenti. In «ogni persona conosce qualcuno», il testimone `y` può cambiare al cambiare di `x`. In «qualcuno conosce ogni persona», dobbiamo invece trovare un unico `x` che soddisfi la condizione per tutte le persone `y`.

1.4 Introduzione del quantificatore universale, '∀I'

$$\frac{A(x)}{\forall y\, A(y)}\ \forall \mathrm{I}$$

Introduzione del quantificatore universale, ∀I

Per dimostrare `∀ y : Cosa, A y`, introduciamo una cosa arbitraria `x`. Il goal diventa `A x`. Dimostriamo `A x` senza usare proprietà particolari di `x`; possiamo allora concludere che `A` vale per ogni oggetto del dominio.

Nel seguente esempio soltanto il primo `intro` applica `∀I`. I due `intro` successivi introducono le assunzioni delle implicazioni.

```
theorem lecture05_forall_intro
    (Cosa : Type)
    (Persona Curiosa : Cosa → Prop) :
    ∀ y : Cosa, Persona y → Curiosa y → Curiosa y := by
  intro x
  intro hXPersona
  intro hXCuriosa
```

```
  exact hXCuriosa
```

Il primo passo introduce `x : Cosa` come oggetto arbitrario. Il goal diventa `Persona x → Curiosa x → Curiosa x`. Introduciamo quindi le due assunzioni e usiamo esattamente `hXCuriosa`.

1.4.1 La condizione di arbitrarietà

L'oggetto introdotto da `∀I` non può essere una persona particolare scelta per una proprietà speciale. Da `hIpaziaCuriosa : Curiosa ipazia`, per esempio, non possiamo concludere `∀ y : Cosa, Curiosa y`. Dopo `intro x`, il goal sarebbe `Curiosa x`, mentre l'assunzione dimostra soltanto `Curiosa ipazia`.

La condizione formale richiede che la variabile usata nella derivazione non compaia libera in alcuna assunzione ancora aperta. In `Curiosa x`, `x` è libera; in `∀ y : Cosa, Curiosa y`, invece, `y` è legata dal quantificatore. Lean realizza questa condizione tramite lo scope: `intro x` crea una nuova variabile locale all'interno del goal universale.

Se riutilizziamo deliberatamente un nome già presente, Lean distingue comunque i due oggetti. Il tactic state può rinominare la vecchia variabile come `x✝`:

```
x✝ : Cosa
hXCuriosa : Curiosa x✝
x : Cosa
⊢ Curiosa x
```

Il simbolo `✝` segnala soltanto una rinomina automatica. I tipi `Curiosa x✝` e `Curiosa x` restano distinti.

1.5 Eliminazione del quantificatore universale, '∀E'

$$\frac{\forall x\, A(x)}{A(t)}\ \forall\mathrm{E}$$

Eliminazione del quantificatore universale, ∀E

Se abbiamo `∀ x : Cosa, A x`, possiamo scegliere un termine `t` del dominio e applicare la dimostrazione universale a `t`. Otteniamo così `A t`.

Vogliamo dimostrare che Marta legge, data l'assunzione che ogni studentessa legge e che Marta è una studentessa.

```
theorem lecture05_forall_elim
    (Cosa : Type)
    (Studentessa Legge : Cosa → Prop)
    (marta : Cosa)
    (hOgniStudentessaLegge :
      ∀ x : Cosa, Studentessa x → Legge x)
    (hMartaStudentessa : Studentessa marta) :
    Legge marta := by
```

```
  have hSeMartaStudentessaAlloraLegge :=
    hOgniStudentessaLegge marta
  have hMartaLegge :=
    hSeMartaStudentessaAlloraLegge hMartaStudentessa
  exact hMartaLegge
```

Applicando l'assunzione universale a `marta` otteniamo `Studentessa marta → Legge marta`. Applichiamo poi questa implicazione all'assunzione `hMartaStudentessa`.

Il termine scelto non deve essere un nome semplice. Può anche essere costruito da una funzione. Se `madreDi : Cosa → Cosa` e `hOgniCosaCuriosa : ∀ x : Cosa, Curiosa x`, allora `hOgniCosaCuriosa (madreDi ipazia)` dimostra `Curiosa (madreDi ipazia)`. Lean controlla il tipo del termine e gestisce le variabili legate senza catturarle accidentalmente.

1.6 Introduzione del quantificatore esistenziale, '∃I'

$$\frac{A(t)}{\exists x\, A(x)}\ \exists \mathrm{I}$$

Introduzione del quantificatore esistenziale, ∃I

Per dimostrare `∃ x : Cosa, A x` dobbiamo scegliere un testimone `t` e dimostrare `A t`. In Lean usiamo il costruttore `Exists.intro`.

```
theorem lecture05_exists_intro
    (Cosa : Type)
    (Legge : Cosa → Prop)
    (marta : Cosa)
    (hMartaLegge : Legge marta) :
    ∃ x : Cosa, Legge x := by
  apply Exists.intro marta
  exact hMartaLegge
```

Scegliendo `marta` come testimone, il goal esistenziale diventa `Legge marta`. Questa proposizione è già disponibile come assunzione.

Anche il testimone può essere un termine costruito. Per dimostrare `∃ x : Cosa, Curiosa x`, possiamo scegliere `madreDi ipazia`, purché abbiamo una dimostrazione di `Curiosa (madreDi ipazia)`.

1.7 Eliminazione del quantificatore esistenziale, ‘∃E‘

$$\dfrac{\exists x\, A(x) \qquad \begin{array}{c}\overline{A(y)}^{\,1} \\ \vdots \\ B\end{array}}{B}\ \exists\mathrm{E}^{\,1}$$

Eliminazione del quantificatore esistenziale, ∃E

Se abbiamo `∃ x : Cosa, A x`, sappiamo che esiste un testimone, ma non possiamo presupporre quale sia. Per ottenere una conclusione `B`, introduciamo un testimone arbitrario `y` e l’assunzione `A y`; da questa assunzione dimostriamo `B`. La conclusione non deve dipendere dall’identità del testimone.

Qualcuno ha fatto una segnalazione. Ogni segnalazione avvia un’indagine. Vogliamo concludere che inizia un’indagine.

```
theorem lecture05_exists_elim
    (Cosa : Type)
    (HaFattoSegnalazione : Cosa → Prop)
    (IniziaIndagine : Prop)
    (hQualcunoHaFattoSegnalazione :
      ∃ x : Cosa, HaFattoSegnalazione x)
    (hChiFaSegnalazioneAvviaIndagine :
      (x : Cosa) → HaFattoSegnalazione x → IniziaIndagine) :
    IniziaIndagine := by
  apply Exists.elim hQualcunoHaFattoSegnalazione
  intro y
  intro hYHaFattoSegnalazione
  have hSeYFaSegnalazioneAlloraIniziaIndagine :=
    hChiFaSegnalazioneAvviaIndagine y
  have hIniziaIndagine :=
    hSeYFaSegnalazioneAlloraIniziaIndagine hYHaFattoSegnalazione
  exact hIniziaIndagine
```

`Exists.elim` apre l’esistenziale. Introduciamo il testimone `y` e il fatto che `y` abbia fatto una segnalazione. La `x` nella seconda assunzione è un segnaposto legato; la applichiamo al testimone locale `y`. Otteniamo prima `HaFattoSegnalazione y → IniziaIndagine` e poi `IniziaIndagine`.

La conclusione è fissata prima dell’apertura dell’esistenziale e non contiene libera la variabile locale `y`. Lean impone questa indipendenza attraverso lo scope di `y` e il tipo di `Exists.elim`: il testimone non può uscire dalla derivazione, salvo essere nuovamente racchiuso in un esistenziale.

1.8 Combinare le regole

Le regole dei quantificatori vengono spesso usate insieme. Supponiamo che qualcuno abbia un ombrello e che chiunque abbia un ombrello resti asciutto. Vogliamo

dimostrare che qualcuno resta asciutto.

```
theorem lecture05_exists_elim_combined
    (Cosa : Type)
    (HaOmbrello RestaAsciutta : Cosa → Prop)
    (hQualcunaHaOmbrello : ∃ x : Cosa, HaOmbrello x)
    (hChiHaOmbrelloRestaAsciutta :
      ∀ x : Cosa, HaOmbrello x → RestaAsciutta x) :
    ∃ x : Cosa, RestaAsciutta x := by
  apply Exists.elim hQualcunaHaOmbrello
  intro y
  intro hYHaOmbrello
  apply Exists.intro y
  have hSeYHaOmbrelloAlloraRestaAsciutta :=
    hChiHaOmbrelloRestaAsciutta y
  have hYRestaAsciutta :=
    hSeYHaOmbrelloAlloraRestaAsciutta hYHaOmbrello
  exact hYRestaAsciutta
```

La dimostrazione combina tre passaggi:

1. `Exists.elim` elimina il primo esistenziale e rende disponibile un testimone `y` con `HaOmbrello y`; 2. l'applicazione di `hChiHaOmbrelloRestaAsciutta` a `y` usa `∀E`; 3. `Exists.intro y` usa lo stesso oggetto come testimone del nuovo esistenziale.

1.8.1 Quantificatore universale e congiunzione

Ogni studentessa che legge prende appunti e comprende. Introduciamo una studentessa arbitraria, applichiamo le due assunzioni universali e costruiamo la congiunzione richiesta.

```
theorem lecture05_forall_conjunction_combined
    (Cosa : Type)
    (Studentessa Legge PrendeAppunti Comprende : Cosa → Prop)
    (hChiLeggePrendeAppunti :
      ∀ x : Cosa, Studentessa x → Legge x → PrendeAppunti x)
    (hChiLeggeComprende :
      ∀ x : Cosa, Studentessa x → Legge x → Comprende x) :
    ∀ x : Cosa,
      Studentessa x → Legge x → PrendeAppunti x ∧ Comprende x := by
  intro x
  intro hXStudentessa
  intro hXLegge
  apply And.intro
  · have hSeXStudentessaAlloraLeggeImplicaAppunti :=
      hChiLeggePrendeAppunti x
    have hSeXLeggeAlloraPrendeAppunti :=
      hSeXStudentessaAlloraLeggeImplicaAppunti hXStudentessa
    have hXPrendeAppunti := hSeXLeggeAlloraPrendeAppunti hXLegge
    exact hXPrendeAppunti
  · have hSeXStudentessaAlloraLeggeImplicaComprende :=
      hChiLeggeComprende x
    have hSeXLeggeAlloraComprende :=
```

```
      hSeXStudentessaAlloraLeggeImplicaComprende hXStudentessa
    have hXComprende := hSeXLeggeAlloraComprende hXLegge
    exact hXComprende
```

1.8.2 Esistenziale e disgiunzione

Qualcuno visita Roma oppure Atene. Apriamo l'esistenziale e consideriamo i due casi della disgiunzione. In entrambi otteniamo un testimone che vede un museo.

```
theorem lecture05_exists_or_combined
    (Cosa : Type)
    (VisitaRoma VisitaAtene VedeMuseo : Cosa → Prop)
    (hQualcunoVisitaUnaCitta :
      ∃ x : Cosa, VisitaRoma x ∨ VisitaAtene x)
    (hChiVisitaRomaVedeMuseo :
      ∀ x : Cosa, VisitaRoma x → VedeMuseo x)
    (hChiVisitaAteneVedeMuseo :
      ∀ x : Cosa, VisitaAtene x → VedeMuseo x) :
    ∃ x : Cosa, VedeMuseo x := by
  apply Exists.elim hQualcunoVisitaUnaCitta
  intro y
  intro hYVisitaRomaOVisitaAtene
  cases hYVisitaRomaOVisitaAtene with
  | inl hYVisitaRoma =>
      apply Exists.intro y
      have hSeYVisitaRomaAlloraVedeMuseo := hChiVisitaRomaVedeMuseo y
      have hYVedeMuseo := hSeYVisitaRomaAlloraVedeMuseo hYVisitaRoma
      exact hYVedeMuseo
  | inr hYVisitaAtene =>
      apply Exists.intro y
      have hSeYVisitaAteneAlloraVedeMuseo := hChiVisitaAteneVedeMuseo y
      have hYVedeMuseo := hSeYVisitaAteneAlloraVedeMuseo hYVisitaAtene
      exact hYVedeMuseo
```

1.8.3 Quantificatori annidati

Il testimone esistenziale può dipendere dalla persona introdotta dal quantificatore universale. Apriamo l'esistenziale soltanto dopo aver fissato quella persona.

```
theorem lecture05_nested_quantifiers_combined
    (Cosa : Type)
    (Persona : Cosa → Prop)
    (Conosce : Cosa → Cosa → Prop)
    (hOgniPersonaConosceQualcuno :
      ∀ x : Cosa, Persona x →
        ∃ y : Cosa, Persona y ∧ Conosce x y) :
    ∀ x : Cosa, Persona x →
      ∃ y : Cosa, Conosce x y ∧ Persona y := by
  intro x
  intro hXPersona
  have hSeXPersonaAlloraConosceQualcuno :=
```

```
    hOgniPersonaConosceQualcuno x
  have hXConosceQualcuno :=
    hSeXPersonaAlloraConosceQualcuno hXPersona
  apply Exists.elim hXConosceQualcuno
  intro y
  intro hYPersonaEXConosceY
  apply Exists.intro y
  apply And.intro
  · exact hYPersonaEXConosceY.right
  · exact hYPersonaEXConosceY.left
```

1.8.4 Esistenziale e negazione

Un testimone ha consegnato ma non ha ricevuto conferma. La regola universale produce la conferma per lo stesso testimone, e la negazione produce `False`.

```
theorem lecture05_exists_negation_combined
    (Cosa : Type)
    (HaConsegnato HaRicevutoConferma : Cosa → Prop)
    (hChiHaConsegnatoRiceveConferma :
      ∀ x : Cosa, HaConsegnato x → HaRicevutoConferma x)
    (hControesempio :
      ∃ x : Cosa, HaConsegnato x ∧ ¬HaRicevutoConferma x) :
    False := by
  apply Exists.elim hControesempio
  intro y
  intro hYHaConsegnatoENonHaRicevutoConferma
  have hYHaConsegnato := hYHaConsegnatoENonHaRicevutoConferma.left
  have hYNonHaRicevutoConferma :=
    hYHaConsegnatoENonHaRicevutoConferma.right
  have hSeYHaConsegnatoAlloraRiceveConferma :=
    hChiHaConsegnatoRiceveConferma y
  have hYHaRicevutoConferma :=
    hSeYHaConsegnatoAlloraRiceveConferma hYHaConsegnato
  have hContraddizione :=
    hYNonHaRicevutoConferma hYHaRicevutoConferma
  exact hContraddizione
```

1.8.5 Esistenziale ed equivalenza

Apriamo l'esistenziale, istanziamo l'equivalenza sul testimone ed estraiamo la direzione necessaria con `Iff.mp`. Lo stesso oggetto testimonia la conclusione.

```
theorem lecture05_exists_iff_combined
    (Cosa : Type)
    (Quadrato HaQuattroLatiUguali : Cosa → Prop)
    (hEquivalenza :
      ∀ x : Cosa, Quadrato x ↔ HaQuattroLatiUguali x)
    (hEsisteQuadrato : ∃ x : Cosa, Quadrato x) :
    ∃ x : Cosa, HaQuattroLatiUguali x := by
  apply Exists.elim hEsisteQuadrato
```

```
  intro y
  intro hYQuadrato
  apply Exists.intro y
  have hEquivalenzaPerY := hEquivalenza y
  have hDirezione := Iff.mp hEquivalenzaPerY
  have hYHaQuattroLatiUguali := hDirezione hYQuadrato
  exact hYHaQuattroLatiUguali
```

In tutti gli esempi affrontiamo prima il costruttore esterno del goal, rendiamo espliciti i testimoni locali e poi applichiamo le regole dei connettivi.

1.9 Strategia operativa

Quando incontriamo una formula quantificata, possiamo porci alcune domande.

- Il goal comincia con ∀? Introduciamo un oggetto arbitrario con `intro`.
- Abbiamo un'assunzione universale? Applichiamola a un termine del dominio.
- Il goal comincia con ∃? Scegliamo un testimone con `Exists.intro`.
- Abbiamo un'assunzione esistenziale? Apriamola con `Exists.elim`, senza presupporre l'identità del testimone.
- Sono presenti più quantificatori? Controlliamo attentamente il loro ordine e quali testimoni possono dipendere da quali variabili.

La strategia estende quella usata per i connettivi proposizionali. Guardiamo il costruttore principale del goal per scegliere una regola di introduzione e le assunzioni composte per scegliere una regola di eliminazione.

1.10 Fonti e approfondimenti

- Jeremy Avigad, Robert Y. Lewis e Floris van Doorn, *Logic and Proof*, «Natural Deduction Rules»: https://leanprover.github.io/logic_and_proof_lean3/nd_quickref.html.
- Lean developers, *The Lean Language Reference*, «Quantifiers»: https://lean-lang.org/doc/reference/latest/Basic-Propositions/Quantifiers/.
- Jeremy Avigad, Robert Y. Lewis e Floris van Doorn, *Logic and Proof*, «First Order Logic in Lean»: https://leanprover.github.io/logic_and_proof_lean3/first_order_logic_in_lean.html.
- Jeremy Avigad, Robert Y. Lewis e Floris van Doorn, *Logic and Proof*, «Natural Deduction for First Order Logic»: https://leanprover.github.io/logic_and_proof_lean3/natural_deduction_for_first_order_logic.html.
- Jeremy Avigad, Leonardo de Moura, Soonho Kong e Sebastian Ullrich, *Theorem Proving in Lean 4*, «Classical Logic»: https://lean-lang.org/theorem_proving_in_lean4/Propositions-and-Proofs/.