Lezione 3: Dalla deduzione naturale alle dimostrazioni in Lean

Stefano M. Nicoletti

1 Lezione 3: Dalla deduzione naturale alle dimostrazioni in Lean

1.1 Registrazione della lezione

Registrazione video della Lezione 3

1.2 Scopo della lezione

Nelle lezioni precedenti abbiamo imparato a leggere un proof state e abbiamo usato tattiche come `intro`, `apply`, `have`, `exact` e `cases`. In questa lezione colleghiamo quei comandi alle regole della deduzione naturale. Vedremo come le tattiche non siano mosse arbitrarie: costruiscono e usano dimostrazioni secondo la struttura logica delle proposizioni.

Il percorso della lezione è il seguente:

- riprendere l'idea di tipo abitato;
- presentare l'idea della corrispondenza di Curry-Howard;
- leggere le regole di introduzione e di eliminazione della deduzione naturale;
- riconoscere le stesse regole nelle dimostrazioni Lean.

1.3 Tipi e abitanti

Un tipo descrive una classe di possibili oggetti. Dire che un tipo è *abitato* significa dire che esiste almeno un termine che ha quel tipo. Se `t` ha tipo `T`, scriviamo:

```
t : T
```

Possiamo leggere questa espressione come «`t` è un termine di tipo `T`» oppure «`t` abita `T`». L'idea non riguarda soltanto Lean. Per esempio, il tipo informale «luna di Giove» è abitato perché Io è una luna di Giove. Io è un oggetto concreto che soddisfa la descrizione espressa dal tipo (Fonte: NASA Science, Io: Facts).

In Lean possiamo osservare la stessa struttura con oggetti più semplici:

```
#check 4       -- 4 : Nat
#check True    -- True : Prop
#check True.intro -- True.intro : True
```

`Nat` è abitato, per esempio, dal termine `4`. Anche `True` è abitato: il termine `True.intro` ha esattamente tipo `True`. Questa seconda osservazione ci porta dalle relazioni generiche tra termini e tipi alle relazioni tra dimostrazioni e proposizioni.

1.4 La corrispondenza di Curry-Howard

Secondo la corrispondenza di Curry-Howard, proposizioni e tipi possono essere letti come due descrizioni della stessa struttura. Una proposizione indica che cosa deve essere dimostrato, mentre il tipo corrispondente indica quale genere di termine deve essere costruito. Una dimostrazione della proposizione è un termine che abita quel tipo (Fonte: Wadler, Propositions as Types).

Ad esempio:

Logica	Teoria dei tipi
proposizione	tipo
dimostrazione	termine
`A → B`	funzione da dimostrazioni di `A` a dimostrazioni di `B`
introduzione di →	costruzione di una funzione
eliminazione di →	applicazione di una funzione

Perciò verificare una dimostrazione significa controllare che il termine costruito abbia il tipo dichiarato. Una derivazione non si limita a certificare in modo astratto che una proposizione è vera: specifica come costruirne una dimostrazione (Fonte: Theorem Proving in Lean 4, Propositions and Proofs).

1.5 Deduzione naturale e proof state

La deduzione naturale presenta una dimostrazione come una sequenza di passi locali governati da regole. Ogni regola mostra come ottenere una conclusione a partire da alcune premesse. Le regole di *introduzione* spiegano come costruire una dimostrazione il cui connettivo principale è quello considerato; le regole di *eliminazione* spiegano come usare una dimostrazione composta già disponibile (Fonte: Logic and Proof, Natural Deduction for Propositional Logic).

Un giudizio ha la forma $\Gamma \vdash A$.

`Γ` è il contesto delle assunzioni disponibili e `A` è la conclusione che vogliamo derivare. Il proof state di Lean riproduce questa organizzazione: il contesto compare sopra il simbolo ⊢, mentre il goal compare dopo di esso.

Quando una regola introduce un'assunzione temporanea, questa viene aggiunta al contesto soltanto nel ramo in cui serve. Per esempio, per dimostrare `A → B` possiamo assumere temporaneamente `A` e costruire `B`. Quando la dimostrazione dell'implicazione è completa, l'assunzione è stata incorporata nella funzione che trasforma dimostrazioni di `A` in dimostrazioni di `B`.

1.6 Implicazione

1.6.1 Introduzione dell'implicazione, '→I'

$$\dfrac{\dfrac{}{A}\,1}{\vdots}\\ \dfrac{B}{A \to B}\,1\ \to$$

Introduzione dell'implicazione, →I

Per dimostrare `A → B`, introduciamo un'assunzione di tipo `A` e costruiamo una dimostrazione di `B`. In Lean, `intro` esegue precisamente questo passaggio.

Vogliamo dimostrare che, se piove e fa freddo, allora piove. Introduciamo l'assunzione che piova e faccia freddo; abbiamo che piove prendendo la parte sinistra dell'assunzione; usiamo esattamente questo fatto.

```
theorem lecture03_imp_intro
    (Piove FaFreddo : Prop) :
    Piove ∧ FaFreddo → Piove := by
  intro hPioveEFaFreddo
  have hPiove := hPioveEFaFreddo.left
  exact hPiove
```

1.6.2 Eliminazione dell'implicazione, '→E'

$$\dfrac{A \to B \qquad A}{B}\ \to\mathrm{E}$$

Eliminazione dell'implicazione, →E

Se abbiamo `A → B` e abbiamo `A`, possiamo applicare la prima dimostrazione alla seconda e ottenere `B`. Questa è di nuovo la regola del *modus ponens*.

Vogliamo dimostrare che prendo l'ombrello, date le assunzioni che, se piove, prendo l'ombrello e che piove. Applichiamo la prima assunzione alla seconda; abbiamo che prendo l'ombrello e usiamo esattamente questo fatto.

```
theorem lecture03_imp_elim
    (Piove PrendoOmbrello : Prop)
    (hPioveOmbrello : Piove → PrendoOmbrello)
    (hPiove : Piove) :
    PrendoOmbrello := by
  have hPrendoOmbrello := hPioveOmbrello hPiove
  exact hPrendoOmbrello
```

Qui `hPioveOmbrello` si comporta come una funzione: riceve il termine `hPiove` di tipo `Piove` e restituisce un termine di tipo `PrendoOmbrello`.

1.7 Congiunzione

1.7.1 Introduzione della congiunzione, '∧I'

$$\frac{A \qquad B}{A \wedge B}\ \wedge\mathrm{I}$$

Introduzione della congiunzione, ∧I

Per dimostrare `A ∧ B` dobbiamo costruire entrambe le componenti. Applicando `And.intro`, il goal viene diviso in due casi: prima `A`, poi `B`.

```
theorem lecture03_and_intro
    (Piove FaFreddo : Prop)
    (hPiove : Piove)
    (hFaFreddo : FaFreddo) :
    Piove ∧ FaFreddo := by
  apply And.intro
  · exact hPiove
  · exact hFaFreddo
```

I due punti elenco (·) appartengono a due goal distinti. Una dimostrazione della congiunzione è completa soltanto quando entrambi sono chiusi (simboleggiano i due congiunti).

1.7.2 Eliminazione della congiunzione, '∧E_l' e '∧E_r'

$$\frac{A \wedge B}{A}\ \wedge\mathrm{E_l}$$

Eliminazione sinistra della congiunzione, ∧E_l

$$\frac{A \wedge B}{B}\ \wedge\mathrm{E_r}$$

Eliminazione destra della congiunzione, ∧E_r

Una dimostrazione di `A ∧ B` contiene una dimostrazione di `A` e una di `B`. Possiamo selezionare la componente sinistra con `.left` e quella destra con `.right`.

```
theorem lecture03_and_elim_left
    (Piove FaFreddo : Prop)
    (hPioveEFaFreddo : Piove ∧ FaFreddo) :
    Piove := by
  have hPiove := hPioveEFaFreddo.left
  exact hPiove
```

```
theorem lecture03_and_elim_right
    (Piove FaFreddo : Prop)
    (hPioveEFaFreddo : Piove ∧ FaFreddo) :
    FaFreddo := by
  have hFaFreddo := hPioveEFaFreddo.right
  exact hFaFreddo
```

L'introduzione costruisce una congiunzione a partire dalle componenti; l'eliminazione percorre il cammino opposto e recupera una componente dalla congiunzione disponibile.

1.8 Disgiunzione

1.8.1 Introduzione della disgiunzione, '∨Iₗ' e '∨Iᵣ'

$$\frac{A}{A \vee B}\ \vee \mathrm{I_l}$$

Introduzione sinistra della disgiunzione, ∨Iₗ

$$\frac{B}{A \vee B}\ \vee \mathrm{I_r}$$

Introduzione destra della disgiunzione, ∨Iᵣ

Per dimostrare `A ∨ B` è sufficiente costruire uno dei due lati, ma dobbiamo specificare quale. `Or.inl` sceglie il lato sinistro; `Or.inr` sceglie il lato destro.

```
theorem lecture03_or_intro_left
    (Piove Nevica : Prop)
    (hPiove : Piove) :
    Piove ∨ Nevica := by
  apply Or.inl
  exact hPiove
```

```
theorem lecture03_or_intro_right
    (Piove Nevica : Prop)
    (hNevica : Nevica) :
    Piove ∨ Nevica := by
  apply Or.inr
  exact hNevica
```

1.8.2 Eliminazione della disgiunzione, '∨E'

$$\dfrac{A \vee B \qquad \begin{array}{c}\overline{A}\,{\scriptstyle 1}\\ \vdots \\ C\end{array} \qquad \begin{array}{c}\overline{B}\,{\scriptstyle 1}\\ \vdots \\ C\end{array}}{C}\,{\scriptstyle 1}\ \vee\mathrm{E}$$

Eliminazione della disgiunzione, ∨E

Se abbiamo `A ∨ B`, non sappiamo in generale quale lato sia stato dimostrato. Per ottenere una stessa conclusione `C`, dobbiamo quindi dimostrarla sia nel caso `A` sia nel caso `B`.

Vogliamo dimostrare che prendo l'ombrello, date le assunzioni che piove oppure nevica, che se piove prendo l'ombrello e che se nevica prendo l'ombrello. Procediamo per casi sulla disgiunzione. In ogni ramo compare soltanto l'assunzione corrispondente a quel caso.

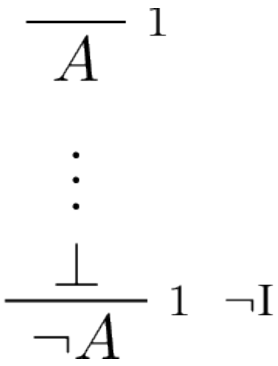
```
theorem lecture03_or_elim
    (Piove Nevica PrendoOmbrello : Prop)
    (hPioveONevica : Piove ∨ Nevica)
    (hPioveOmbrello : Piove → PrendoOmbrello)
    (hNevicaOmbrello : Nevica → PrendoOmbrello) :
    PrendoOmbrello := by
  cases hPioveONevica with
  | inl hPiove =>
      have hPrendoOmbrello := hPioveOmbrello hPiove
      exact hPrendoOmbrello
  | inr hNevica =>
      have hPrendoOmbrello := hNevicaOmbrello hNevica
      exact hPrendoOmbrello
```

`cases` elimina la disgiunzione rendendo esplicite le varie possibilità tramite una dimostrazione per casi.

1.9 Negazione, verità e falsità

1.9.1 Negazione

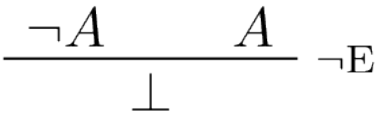
In Lean, `¬A` è definita come `A → False`. Una dimostrazione di `¬A` è quindi una funzione che trasforma ogni ipotetica dimostrazione di `A` in una contraddizione.



Introduzione della negazione, ¬I

Per introdurre `¬A`, introduciamo l'assunzione `A` e costruiamo `False`. Vogliamo dimostrare che non ho bevuto caffè, date le assunzioni che, se bevo caffè, resto sveglio e che non resto sveglio.

```
theorem lecture03_not_intro
    (BevoCaffe RestoSveglio : Prop)
    (hCaffeSveglio : BevoCaffe → RestoSveglio)
    (hNonSveglio : ¬RestoSveglio) :
    ¬BevoCaffe := by
  intro hBevoCaffe
  have hRestoSveglio := hCaffeSveglio hBevoCaffe
  have hContraddizione : False := hNonSveglio hRestoSveglio
  exact hContraddizione
```



Eliminazione della negazione, ¬E

Per eliminare una negazione, applichiamo `¬A` ad `A` e otteniamo `False`.

```
theorem lecture03_not_elim
    (LaboratorioAperto : Prop)
    (hLaboratorioAperto : LaboratorioAperto)
    (hLaboratorioNonAperto : ¬LaboratorioAperto) :
```

```
    False := by
  have hContraddizione :=
    hLaboratorioNonAperto hLaboratorioAperto
  exact hContraddizione
```

1.9.2 Verità e falsità

$$\frac{}{\top}\ \top\mathrm{I}$$

Introduzione della verità, $\top I$

`True` ha un costruttore canonico e non richiede assunzioni:

```
theorem lecture03_true_intro : True := by
  exact True.intro
```

`exact True.intro` può anche essere sostituito con la tattica `trivial`.

$$\frac{\bot}{A}\ \bot\mathrm{E}$$

Eliminazione della falsità, $\bot E$

`False`, invece, non ha costruttori. Se il contesto contiene già una dimostrazione di `False`, possiamo eliminare la falsità e ottenere qualunque proposizione. Questo principio è chiamato anche *ex falso quodlibet*, dal falso segue qualsiasi proposizione: `apply False.elim` può anche essere sostituito con la tattica `exfalso`, infatti.

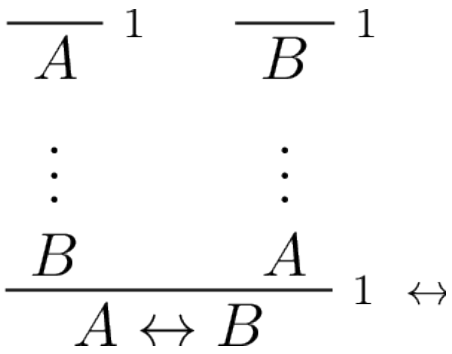
```
theorem lecture03_false_elim
    (Piove : Prop)
    (hContraddizione : False) :
    Piove := by
  apply False.elim
  exact hContraddizione
```

Questa regola non dice che possiamo dimostrare arbitrariamente qualsiasi cosa: possiamo farlo se abbiamo già costruito una contraddizione.

1.10 Bicondizionale

Una dimostrazione di `A ↔ B` contiene due funzioni: una da `A` a `B` e una da `B` ad `A`.

1.10.1 Introduzione del bicondizionale, '↔I'

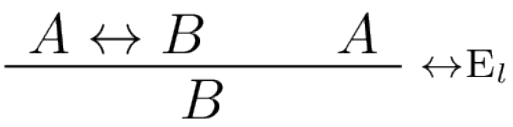


Introduzione del bicondizionale, ↔I

Per dimostrare che «piove e fa freddo» equivale a «fa freddo e piove», applichiamo `Iff.intro`. Nel primo caso costruiamo la direzione da sinistra a destra; nel secondo costruiamo la direzione inversa.

```
theorem lecture03_iff_intro
    (Piove FaFreddo : Prop) :
    Piove ∧ FaFreddo ↔ FaFreddo ∧ Piove := by
  apply Iff.intro
  · intro hPioveEFaFreddo
    apply And.intro
    · exact hPioveEFaFreddo.right
    · exact hPioveEFaFreddo.left
  · intro hFaFreddoEPiove
    apply And.intro
    · exact hFaFreddoEPiove.right
    · exact hFaFreddoEPiove.left
```

1.10.2 Eliminazione del bicondizionale, '↔Eₗ' e '↔Eᵣ'



Eliminazione sinistra-destra del bicondizionale, ↔Eₗ

$$\frac{A \leftrightarrow B \qquad B}{A} \leftrightarrow \mathrm{E}_r$$

Eliminazione destra-sinistra del bicondizionale, $\leftrightarrow E_r$

`Iff.mp` estrae la direzione sinistra-destra (la direzione del `modus ponens`), mentre `Iff.mpr` estrae la direzione destra-sinistra (`modus ponens reversed`).

```
theorem lecture03_iff_elim_left
    (NumeroPari DivisibilePerDue : Prop)
    (hEquivalenza : NumeroPari ↔ DivisibilePerDue)
    (hNumeroPari : NumeroPari) :
    DivisibilePerDue := by
  have hDirezione := Iff.mp hEquivalenza
  have hDivisibilePerDue := hDirezione hNumeroPari
  exact hDivisibilePerDue
```

```
theorem lecture03_iff_elim_right
    (NumeroPari DivisibilePerDue : Prop)
    (hEquivalenza : NumeroPari ↔ DivisibilePerDue)
    (hDivisibilePerDue : DivisibilePerDue) :
    NumeroPari := by
  have hDirezione := Iff.mpr hEquivalenza
  have hNumeroPari := hDirezione hDivisibilePerDue
  exact hNumeroPari
```

Dopo aver estratto la direzione necessaria, la applichiamo come una normale implicazione.

1.11 Come scegliere una regola

Di fronte a un proof state, conviene porsi alcune domande.

La prima riguarda il goal: qual è il suo connettivo principale? Se il goal è un'implicazione, una congiunzione, una disgiunzione, una negazione o un bicondizionale, una regola di introduzione suggerisce come costruirlo.

La seconda riguarda il contesto: quali dimostrazioni composte sono già disponibili? Un'implicazione può essere applicata; una congiunzione può essere scomposta; una disgiunzione richiede casi; una negazione può essere applicata alla proposizione che nega; un bicondizionale offre due direzioni.

Le principali corrispondenze operative sono:

Tattica o termine Lean	Lettura nella deduzione naturale
`intro h`	introduciamo un'assunzione
`apply C`	applichiamo un costruttore o una regola
`have h := ...`	abbiamo un nuovo fatto intermedio
`exact h`	usiamo esattamente il termine richiesto
`cases h with`	abbiamo i diversi casi possibili
`h.left`, `h.right`	eliminiamo una congiunzione
`hAB hA`	eliminiamo un'implicazione

Questa terminologia descrive la struttura del termine che Lean sta costruendo e rende leggibile il rapporto tra regola formale, argomento in linguaggio naturale e proof script.

1.12 Strategia di lavoro

Quando costruiamo una dimostrazione:

1. leggiamo il goal e le assunzioni; 2. identifichiamo il connettivo principale del goal; 3. scegliamo una regola di introduzione, se il goal lo suggerisce; 4. cerchiamo nelle assunzioni una regola di eliminazione utile; 5. rendiamo espliciti con `have` i risultati intermedi importanti; 6. chiudiamo il goal con `exact` quando abbiamo un termine del tipo richiesto; 7. controlliamo che tutti i casi e tutti i sotto-goal siano stati risolti.

Il file `Classroom.lean` mostra ciascuna regola separatamente. Il file `Exercises.lean` propone prima applicazioni dirette e poi argomenti che combinano più regole. Il file `Solutions.lean` mostra come ogni passaggio in Lean possa essere ricondotto alla regola di deduzione naturale corrispondente.

1.13 Fonti e letture

- Jeremy Avigad, Leonardo de Moura, Soonho Kong e Sebastian Ullrich, *Theorem Proving in Lean 4*, capitolo «Propositions and Proofs»: https://lean-lang.org/theorem_proving_in_lean4/Propositions-and-Proofs/.
- 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.
- Philip Wadler, «Propositions as Types», *Communications of the ACM* 58(12), 2015: https://homepages.inf.ed.ac.uk/wadler/papers/propositions-as-types/propositions-as-types.pdf.
- NASA Science, «Io: Facts»: https://science.nasa.gov/jupiter/jupiter-moons/io/facts/.