Logica

Lezione 1

Dr Marco Benini

marco.benini@uninsubria.it

Dipartimento di Scienza e Alta Tecnologia
Università degli Studi dell'Insubria

a.a. 2020/21

Syllabus

Burocrazia:

- Introduzione
- Programma
- Esami
- Orari
- Domande

Una breve introduzione alla logica:

- Di cosa tratta?
- Da dove viene?
- Perché ad Informatica?

Introduzione

La Logica Matematica è il campo della Matematica che studia il processo di dimostrazione e i fondamenti della disciplina.

Questo corso introdurrà la Logica dall'inizio presumendo una conoscenza minima di matematica elementare.

Il materiale del corso è, più o meno, standard e molti testi introduttivi lo coprono. I lucidi contengono tutti gli argomenti richiesti all'esame.

Una particolare attenzione sarà dedicata alla relazione tra Logica e Informatica, fornendo spunti e connessioni con altri corsi.

Programma

Il corso è di 48 ore. Il programma dettagliato è:

1. **Introduzione**: storia della logica, relazione con l'informatica;
2. **Logica Proposizionale**: induzione, sintassi, deduzione naturale, semantica a tavole di verità, insiemi minimali di connettivi, forme normali, soddisfacibilità, algebre di Boole, correttezza e completezza;
3. **Logica al Primo Ordine**: sintassi, sostituzione, calcolo naturale, semantica alla Tarski, correttezza, risoluzione, unificazione, programmazione logica;
4. **Teoria dei Tipi Semplici**: λ-calcolo (sintassi, riduzioni, tipi di dati, programmazione funzionale), teoria dei tipi semplici (sintassi, riduzioni), logica proposizionale intuizionista (sintassi, potenza espressiva, isomorfismo Curry-Howard), dimostrazioni come programmi, correttezza per costruzione, normalizzazione forte, terminazione;
5. **Risultati limitativi**: compattezza, aritmetica di Peano, codifica, lemma di punto fisso, teorema di incompletezza di Gödel, funzioni calcolabili, tesi di Church-Turing, teorema di Cantor, incalcolabilità e incompletezza.

Materiale didattico

I lucidi sono disponibili sul sito web del corso:

`https://marcobenini.me/lectures/logica-informatica/`

Alla fine di ogni lezione, si daranno riferimenti ad articoli, testi, e altre risorse per approfondire. Mentre il contenuto dei lucidi è **richiesto**, il materiale di approfondimento è **opzionale**.

Alcuni video, in inglese ma sottotitolati in italiano, sono disponibili sul sito web del corso.

Sebbene non vi sia un testo ufficiale del corso, userò principalmente il classico Bell, Machover *A Course in Mathematical Logic*, dando indicazione di volta in volta della provenienza del materiale.

Esame

L'esame consiste in una prova scritta in tre parti:

1. (10 punti) due domande relative alla presentazione e applicazione delle nozioni presentate a lezione;
2. (10 punti) dimostrazione di un teorema;
3. (12 punti) due esercizi sulla falsariga di quelli svolti in aula.

L'esame può essere sostenuto in qualsiasi appello **dopo** la fine del corso previa iscrizione.

I criteri di valutazione sono la conoscenza delle nozioni, la comprensione dei concetti, la capacità di applicarli.

Orari

L'orario delle lezioni è fissato e non può essere facilmente modificato. In generale, una lezione inizierà 10 minuti dopo l'orario ufficiale e terminerà 10 minuti prima, così da permettere agli studenti di muoversi tra le aule. Non ci sono pause in mezzo alla lezione.

Domande

Le domande sono benvenute. Non esitate a porre questioni quando qualcosa non è chiaro durante una lezione. Le domande possono anche essere rivolte prima e dopo una lezione, specialmente se non attinenti alla lezione del giorno.

Si possono anche porre domande per email: scrivete all'indirizzo

`marco.benini@uninsubria.it`

specificando il vostro nome, cognome, corso, e la domanda. Usate il vostro indirizzo ufficiale di posta elettronica di Ateneo (`uninsubria`).

Non c'è un orario di ricevimento. Per essere ricevuti dal docente dovete fissare un appuntamento. Fatelo solo se ritenete che non vi sia altro modo per risolvere il vostro problema: sebbene il docente è generalmente reperibile durante il periodo di svolgimento del corso, al di fuori di esso non è garantito che egli sia in sede o anche in Italia. Eventualmente, considerate la possibilità di un incontro in teleconferenza.

Il vostro docente

Sono un ricercatore in Logica Matematica. Questo significa che il mio lavoro principale è pensare e, talora, dimostrare nuovi risultati in questo campo della Matematica.

Insegnare è parte della mia attività accademica, ma non la mia principale occupazione. Oltre a questo corso, insegno anche al corso di Mathematical Logic presso il corso di laurea magistrale in Matematica.

Come matematico, i miei interessi di ricerca vertono principalmente nella relazione tra verità e calcolabilità, in particolare mi occupo di teoria omotopica dei tipi, calcolabilità nei sistemi costruttivi, e teoria strutturale della dimostrazione.

Per saperne di più,

`http://marcobenini.me`

Logica matematica

Logica Matematica

La logica matematica studia il processo di dimostrazione matematica e la nozione di verità, in senso ampio.

La logica è una branca antica del pensiero matematico: le sue origini risalgono ad Aristotele, mentre i fondamenti matematici possono essere tracciati nei lavori di Boole, Frege, Cantor, Russell, Hilbert, Gödel, ...

A partire dal risultato di incompletezza di Gödel, la disciplina ha subito un enorme sviluppo, e oggigiorno è un campo molto attivo della Matematica contemporanea, con importanti applicazioni in Informatica e Filosofia.

Poiché questo è un primo corso in logica matematica, ed è orientato all'informatica, noi introdurremo i risultati principali che sottendono gli sviluppi legati all'informatica, in particolare in relazione ai metodi di programmazione, e rispetto alla calcolabilità.

La matematica greca

©Marco Benini, Pytaghoras in Samos

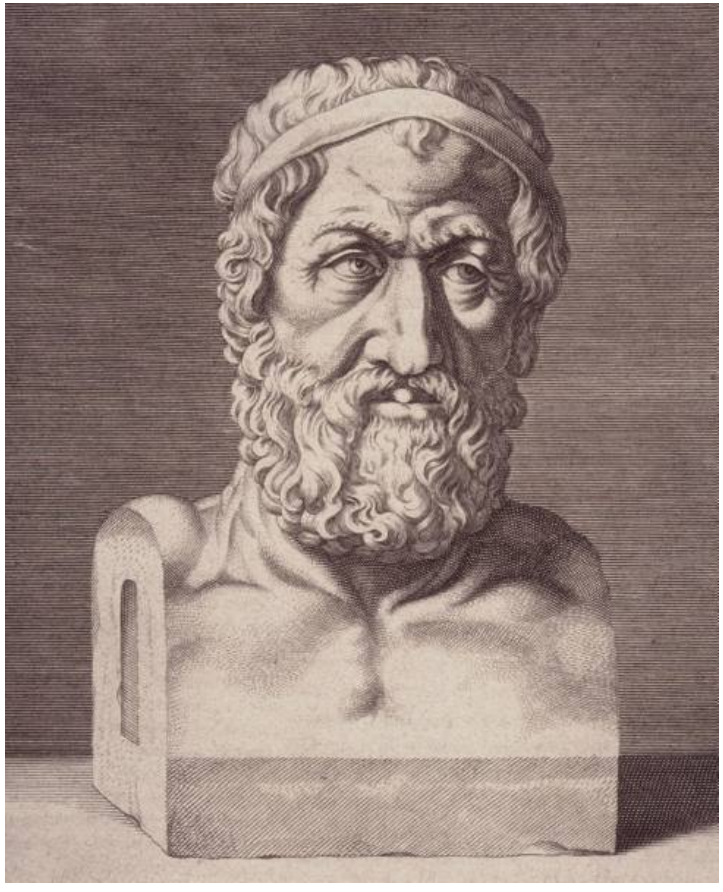
Dimostrazione
Teorema

La matematica greca

Logos

Johannes Moreelse,
Heraclite,
Centraal Museum, Utrecht,
1630

Prova per contraddizione
Paradossi dell'infinito



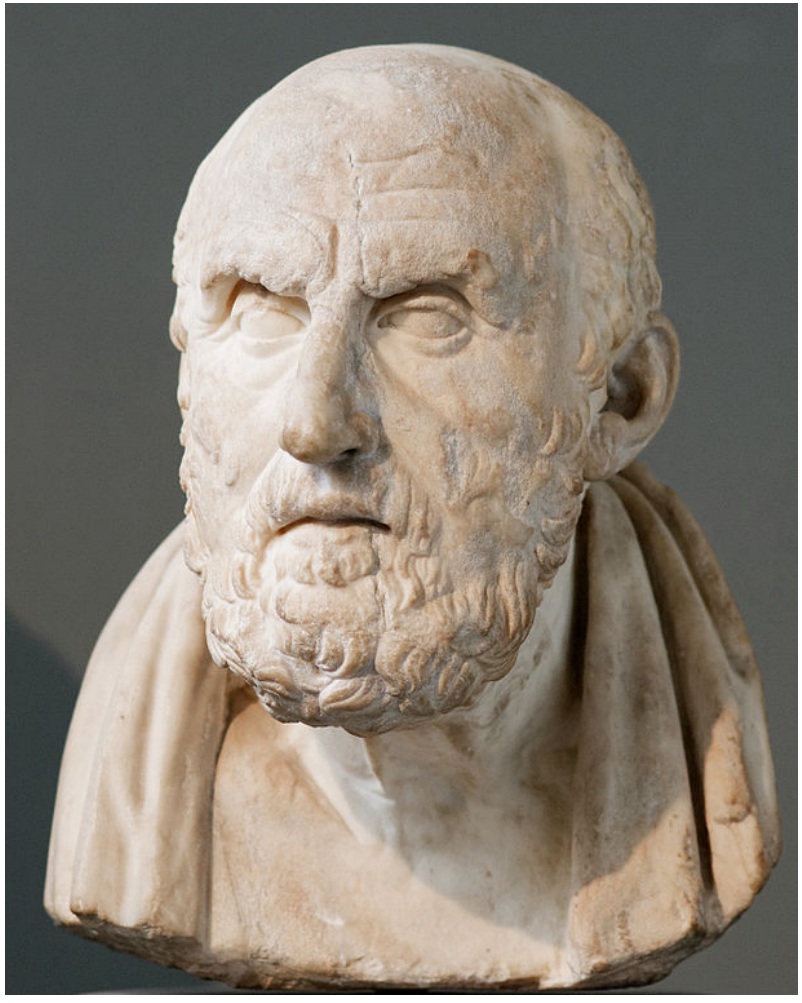
Marcus Meibomius,
engraving of Zeno of Elea in Diogenis Laertii De Vitis,
1698

La matematica greca

Ragionamento formale
Terzo escluso
Sistema formale

Organon,
Aristotle

La matematica greca



Chrysippos of Soli,
Marble, Roman copy of the late 3rd century BC

Modalità
Condizionali, implicazione
Relazione tra significato e verità, semantica

Islam

Avicenna
Portrait on Silver Vase
Museum at BuAli Sina (Avicenna) Mausoleum
Hamadan, Western Iran
©Adam Jones photographer, 2012

Connettivi modali e temporali
Precursori degli oggetti ideali
Algebra
Algoritmo

Europa medievale

Museum of Galileo, Florence,
©Marco Benini, 2015

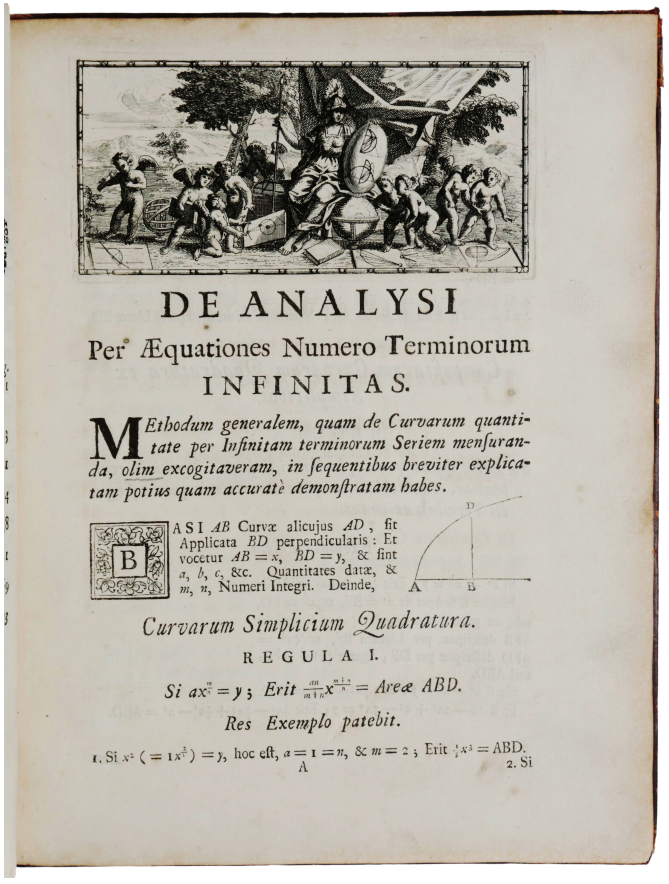
Cartesio

Portrait of René Descartes by Frans Hals

Geometria analitica
Ponte tra geometria e algebra
Spazio

Newton

Analisi matematica
Fondamenti nella geometria euclidea



DE ANALYSI

Per Æquationes Numero Terminorum

INFINITAS.

Methodum generalem, quam de Curvarum quantitate per Infinitam terminorum Seriem menſuranda, olim excogitaveram, in ſequentibus breviter explicatam potius quam accuratè demonſtratam habes.

BASI *AB* Curvæ alicujus *AD*; fit Applicata *BD* perpendicularis: Et vocetur $AB = x$, $BD = y$, & ſint a, b, c, &c. Quantitates datæ, & m, n, Numeri Integri. Deinde,

Curvarum Simplicium Quadratura.

REGULA I.

Si $ax^{\frac{m}{n}} = y$; *Erit* $\frac{an}{m+n}x^{\frac{m+n}{n}} =$ *Areæ ABD.*

Res Exemplo patebit.

1. Si $x^2 (= 1x^{\frac{2}{1}}) = y$, hoc eſt, $\frac{a}{A} = 1 = n$, & $m = 2$; Erit $\frac{1}{3}x^3 =$ ABD. 2. Si

De analysi per aequationes numero terminorum infinitas, Isaac Newton, 1711

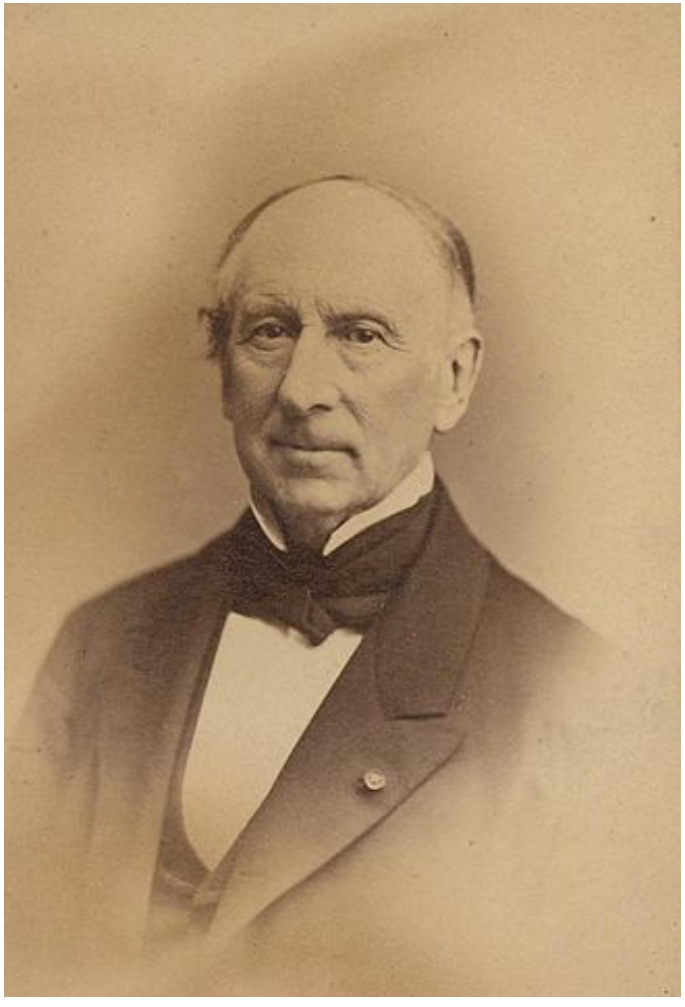
Liebniz

Bildnis des Philosophen Gottfried Wilhelm Freiherr von Leibniz,
Christoph Bernhard Francke, 1695

Analisi matematica
Characteristica Universalis

La crisi dell'analisi

Carl Friedrich Gauß,
Christian Albrecht Jensen, 1840



Augustin Louis Cauchy,
photo by Charles Reutlinger

La crisi dell'analisi

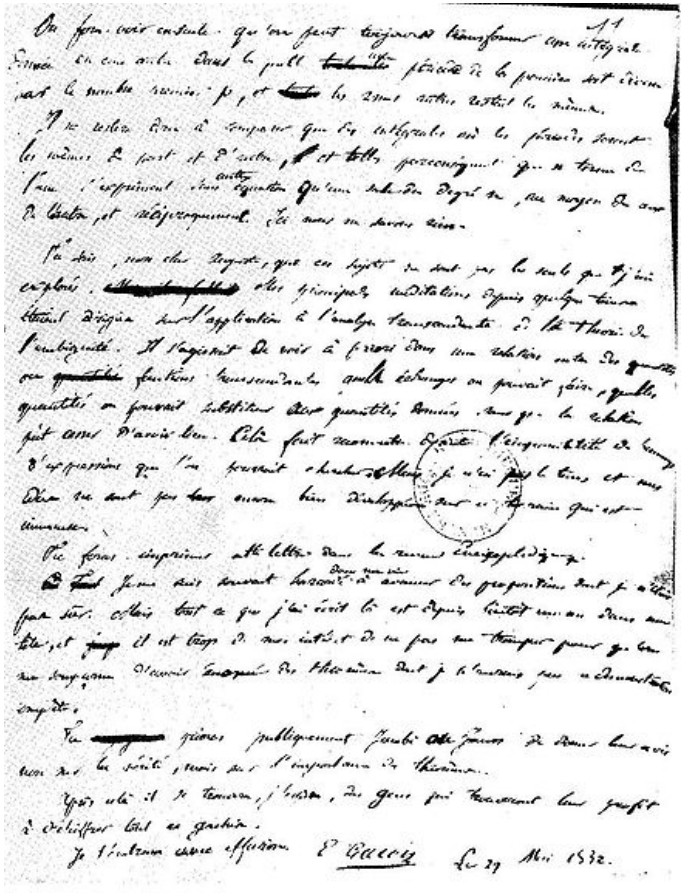
Georg Friedrich Bernhard Riemann, 1863

Julius Wilhelm Richard Dedekind,
photo by Johannes Ganz, 1866

Karl Weierstraß,
Conrad Fehr, 1895

Rivoluzione

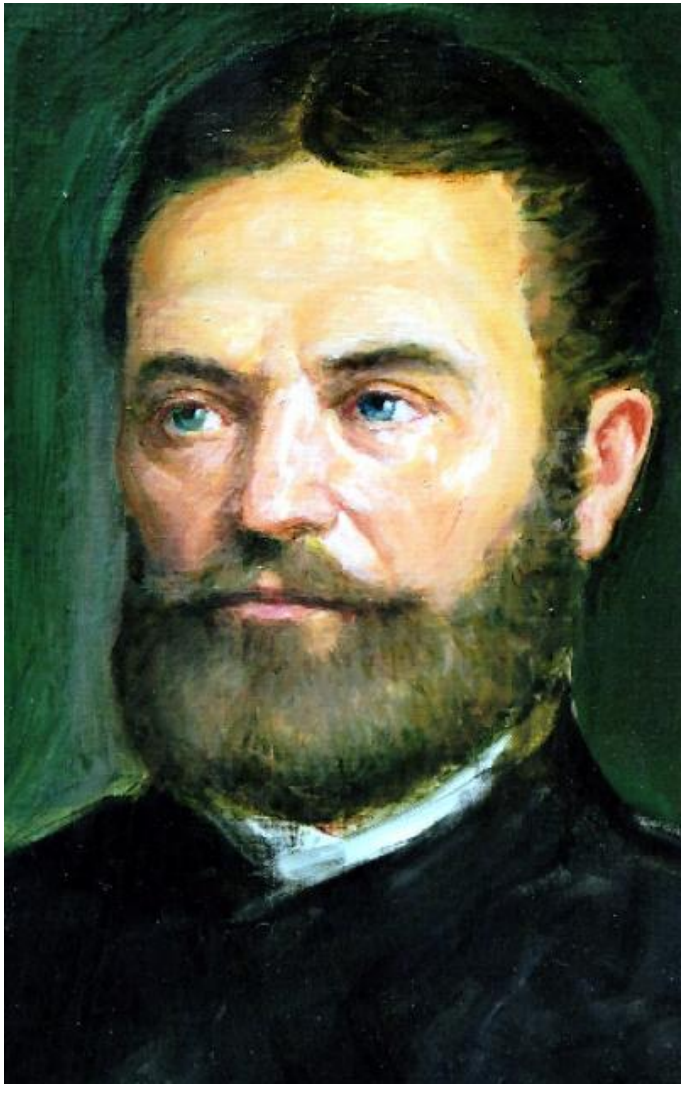
Algebra



last page of the letter from Évariste Galois to Auguste Chevalier, 29th March 1832

Niels Henrik Abel

Geometria non euclidea

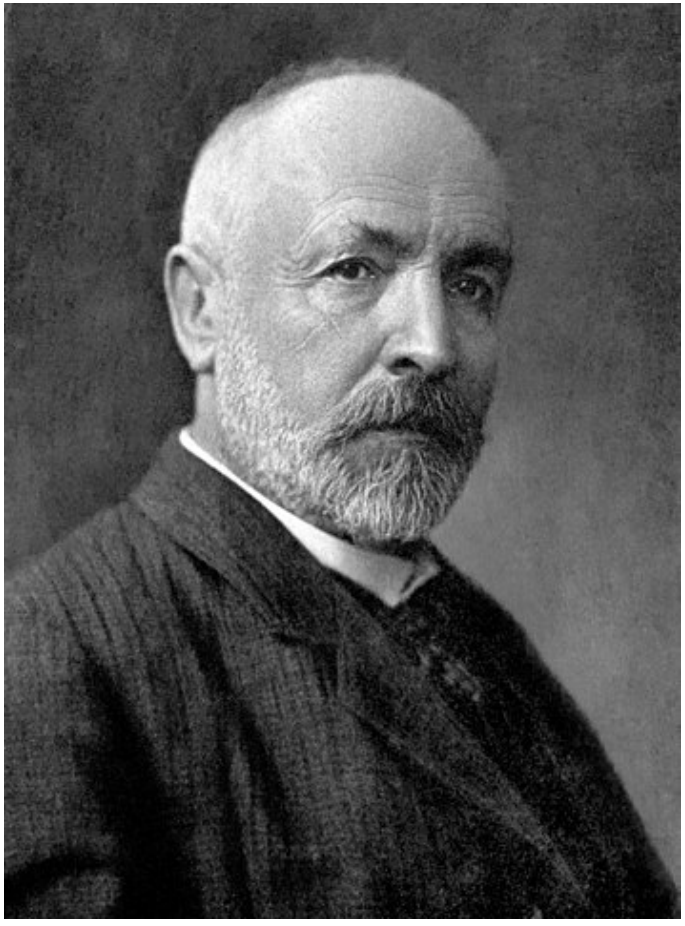


János Bolyai,
painting by Márkos Ferenc, 2012

Nikolai Ivanovich Lobachevsky,
portrait by Lev Kryukov, 1843

Fondamenti della matematica

Cantor

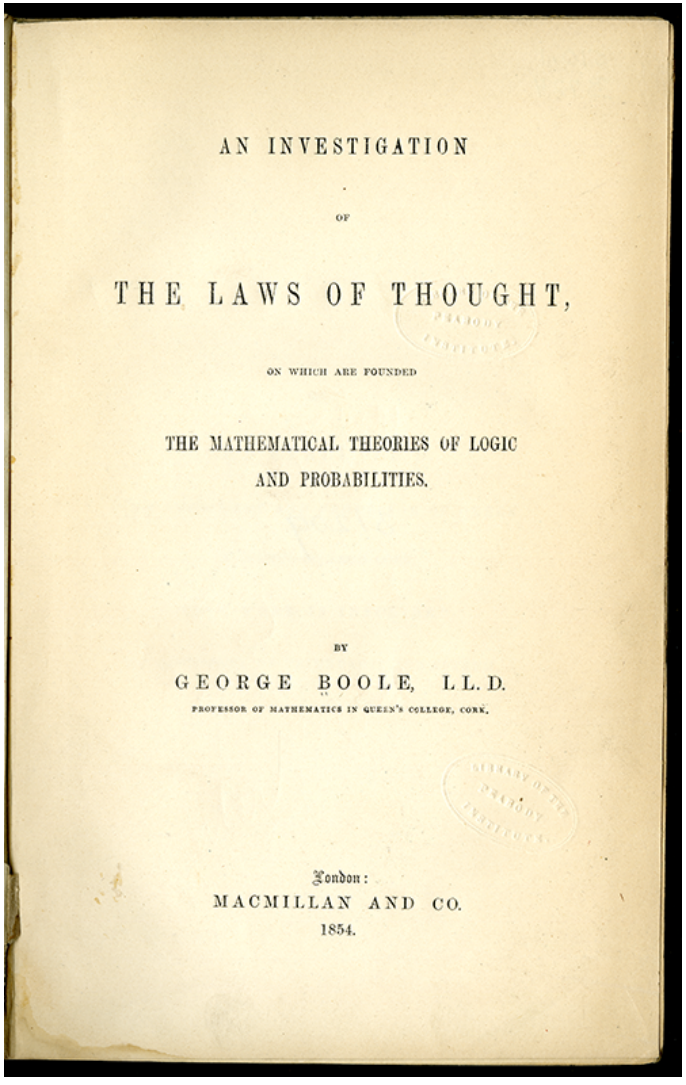


Georg Ferdinand Ludwig Philipp Cantor,
1910

Teoria degli insiemi
Infiniti
Cardinalità

Boole

The Laws of Thought
Logica matematica


AN INVESTIGATION

OF

THE LAWS OF THOUGHT,

ON WHICH ARE FOUNDED

THE MATHEMATICAL THEORIES OF LOGIC
AND PROBABILITIES.

BY

GEORGE BOOLE, LL.D.

PROFESSOR OF MATHEMATICS IN QUEEN'S COLLEGE, CORK.

London:
MACMILLAN AND CO.
1854.

The Laws of Thought,
1854

Frege

Friedrich Ludwig Gottlob Frege,
1879

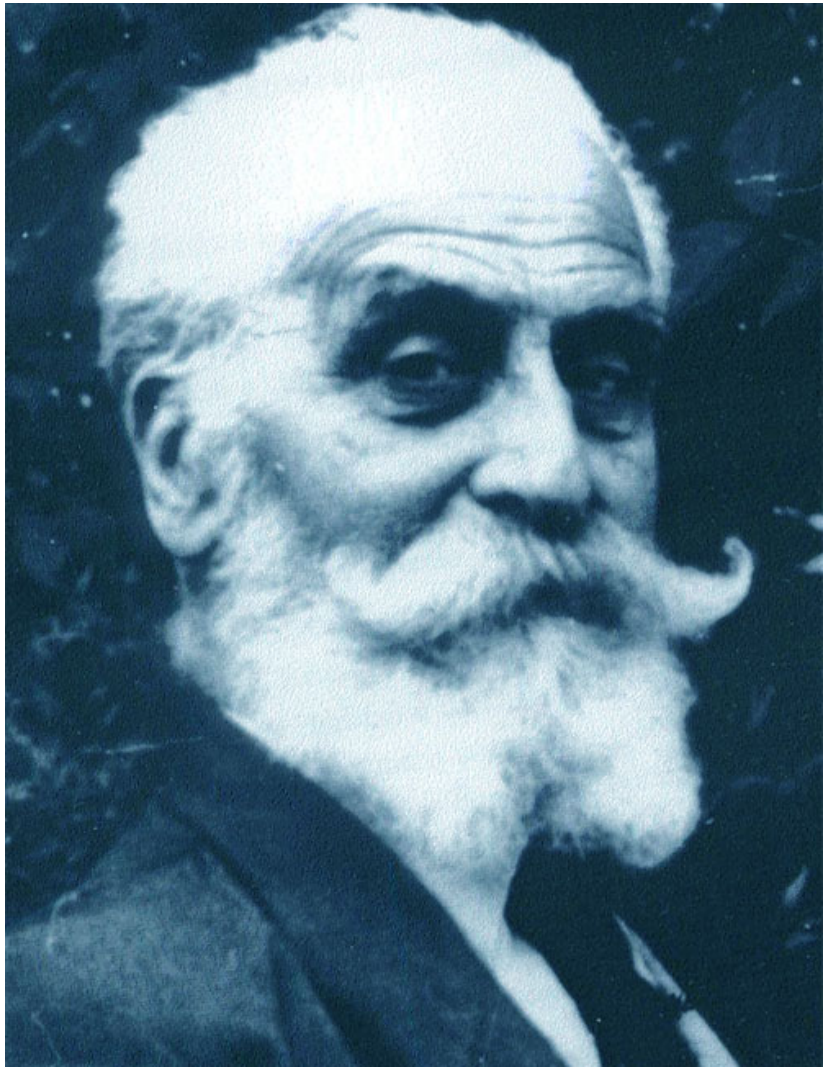
Variabili
Quantificatori

Paradosso di Russell

Sia $R = \{x : x \notin x\}$.

Allora $R \in R$ se e solo se $R \notin R$.

Peano



Giuseppe Peano

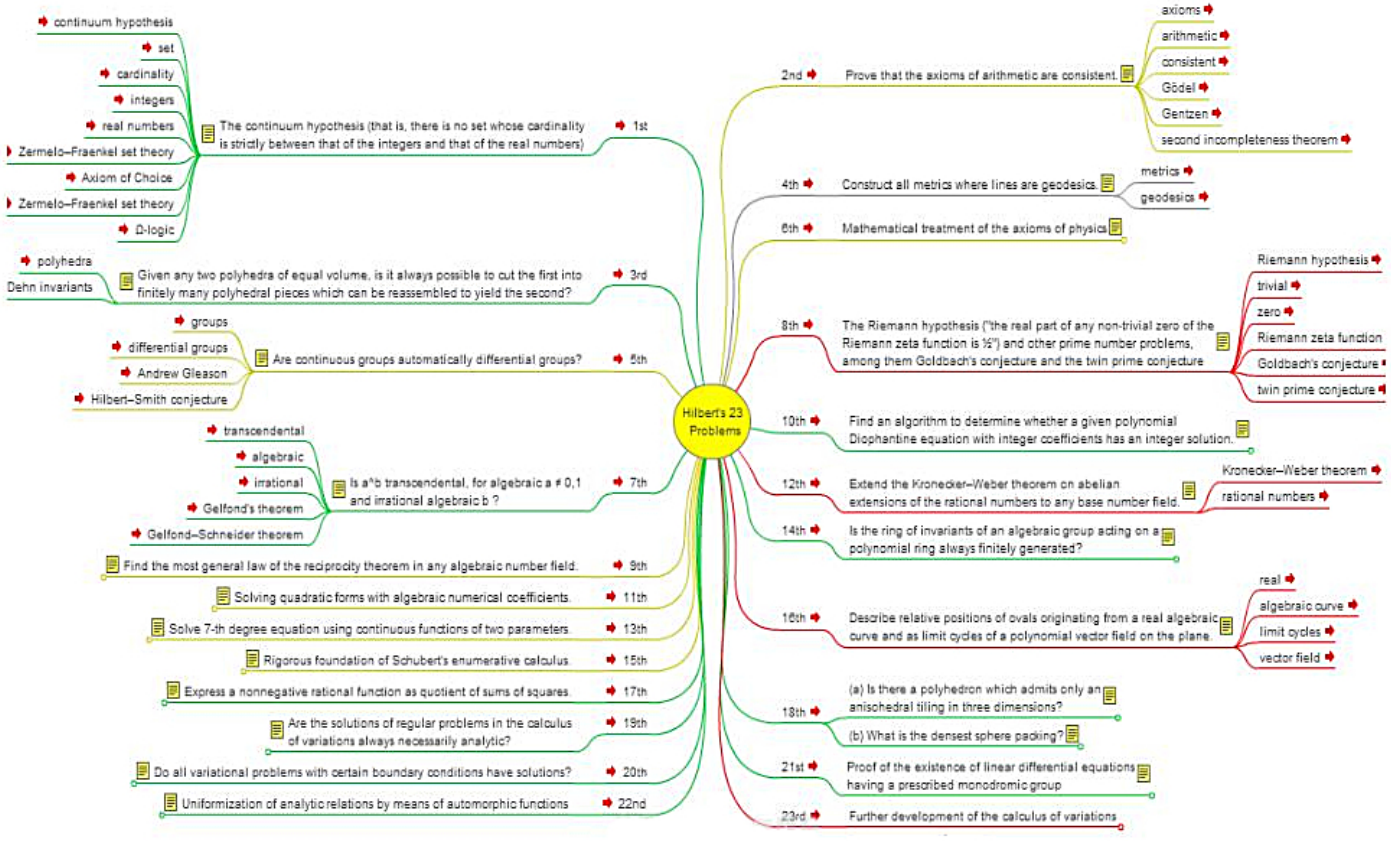
Aritmetica formale
Induzione

Hilbert

David Hilbert,
1912

Geometria formale

Hilbert



© Sharjeel Khan

Hilbert

Formalizzazione: tutte le affermazioni matematiche devono essere scritte, almeno in linea di principio, in un preciso linguaggio formale e devono essere manipolate per mezzo di un insieme di regole fissato, preciso e formale.

Consistenza: occorre dimostrare che l'intero corpus della matematica sia privo di contraddizioni per mezzo di una prova formale all'interno della matematica stessa.

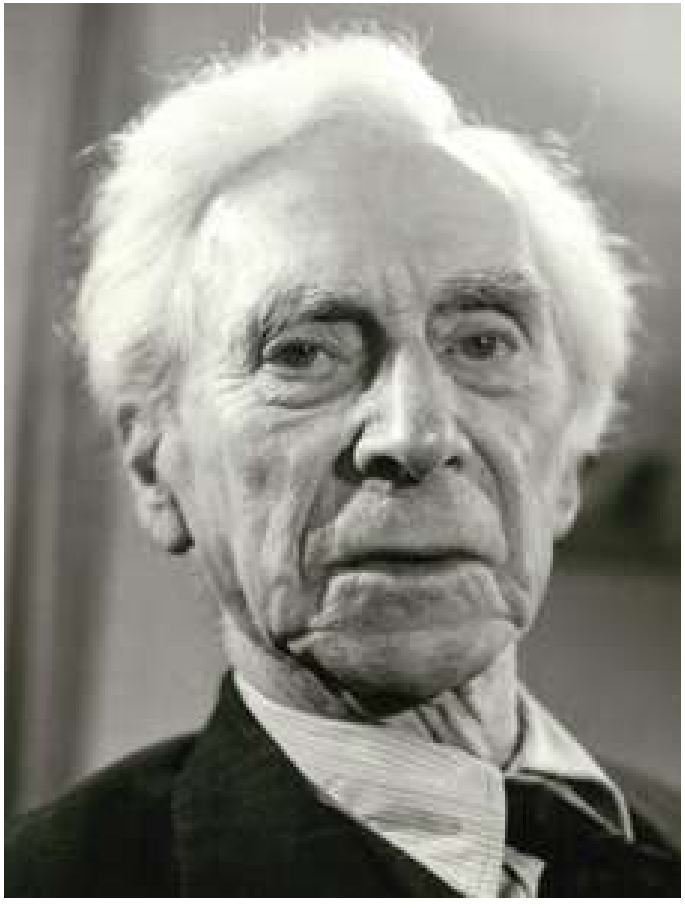
Finitistica: il linguaggio, le regole di inferenza e le dimostrazioni devono essere finite ed effettive. In particolare la prova di consistenza deve essere finitistica.

Zermelo e Frænkel

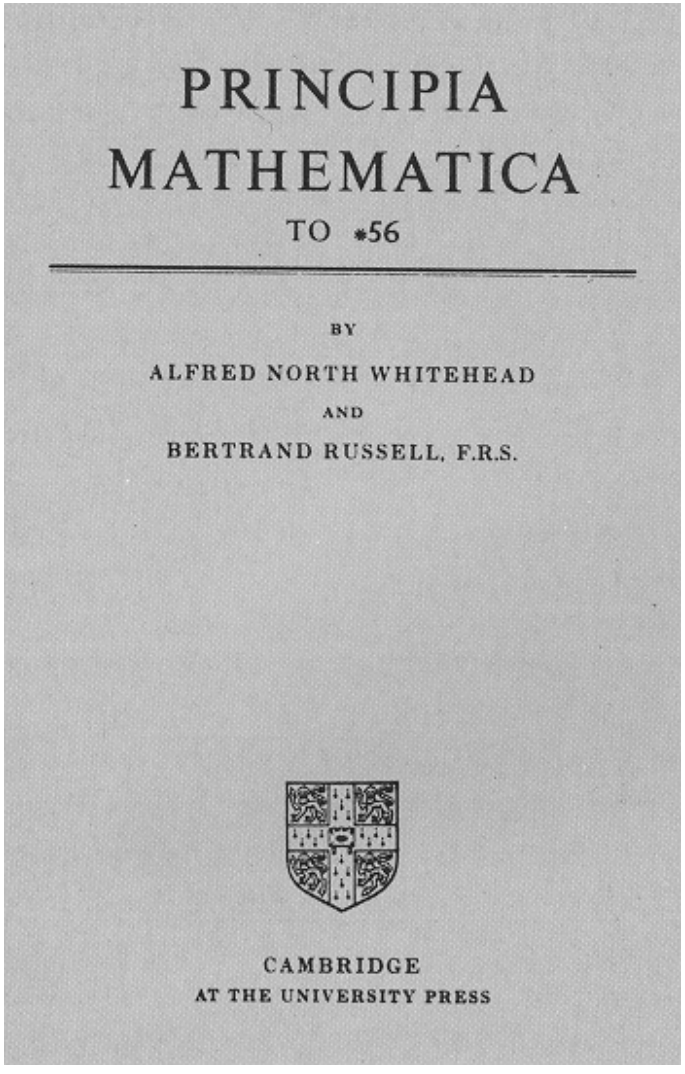
Ernst Zermelo,
1900

Abraham Halevi Frænkel,
1939–49

Russell



Bertrand Russell



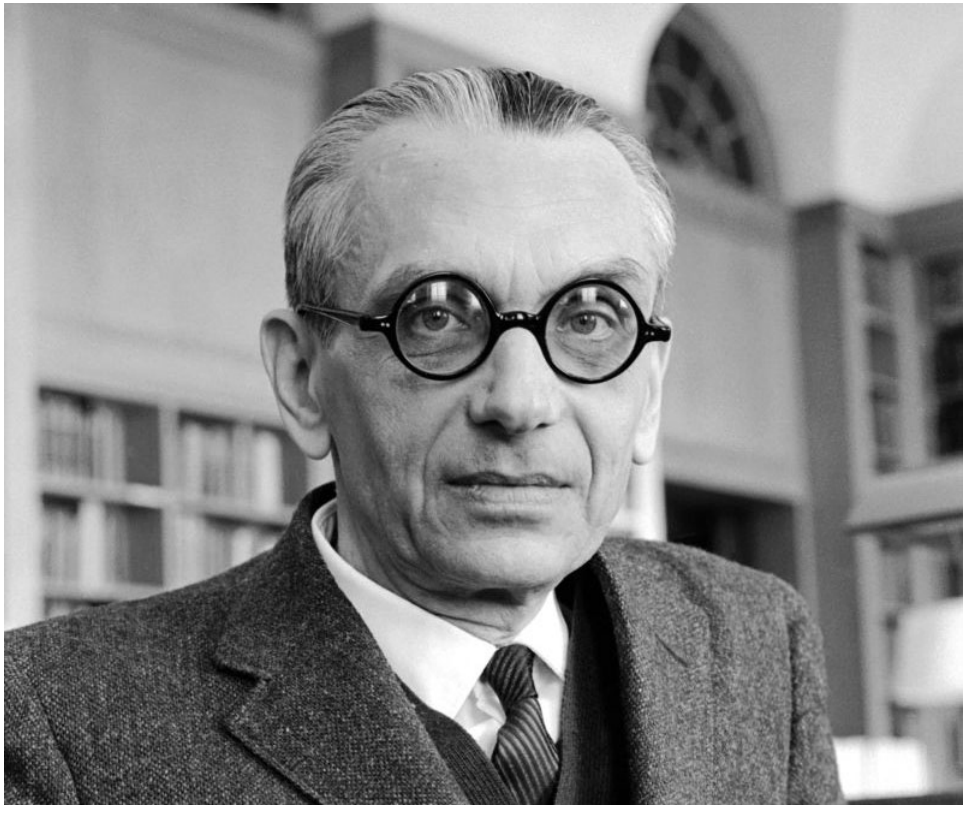
Principia Mathematica

Löwenheim e Skolem

Leopold Löwenheim

Thoralf Skolem,
1930

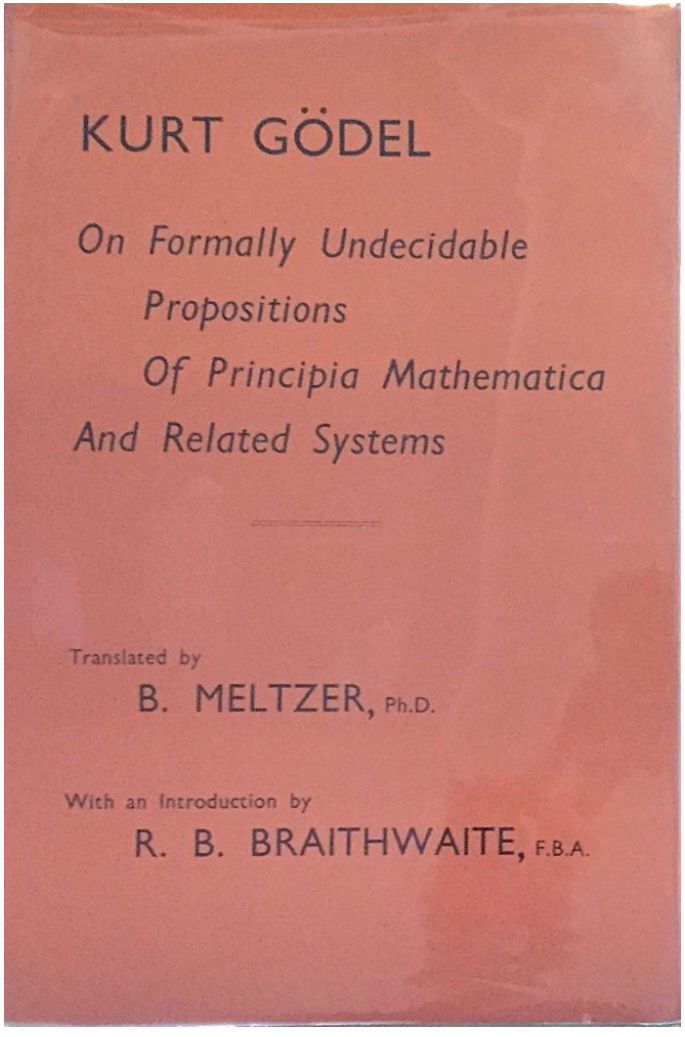
Gödel



Kurt Gödel

Completezza della
logica al primo ordine

Gödel



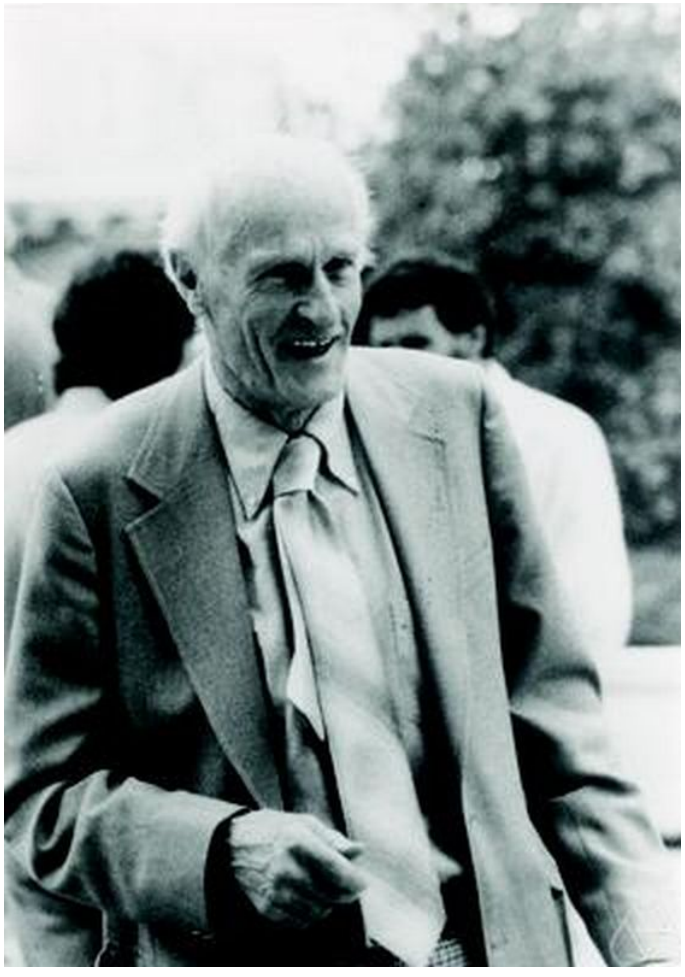
Teoremi di incompletezza

Gentzen

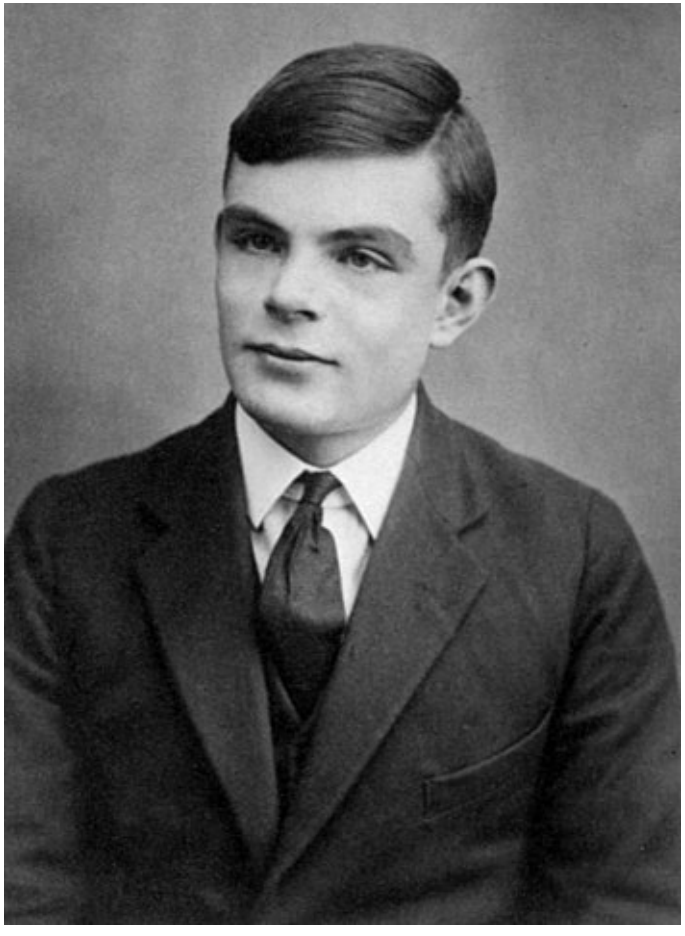
Gerhard Gentzen,
photo by Eckart Menzler-Trott, Prague, 1945

Consistenza dell'aritmetica
Eliminazione del taglio
Teoria della dimostrazione

Teoria della calcolabilità



Stephen Cole Kleene,
photo by Konrad Jacobs, Erlangen, 1978



Alan Mathison Turing

Teoria della calcolabilità

Problema dell'arresto
Tesi di Church-Turing

Alonzo Church

Intuizionismo

Luitzen Egbertus Jan Brouwer

Matematica costruttiva

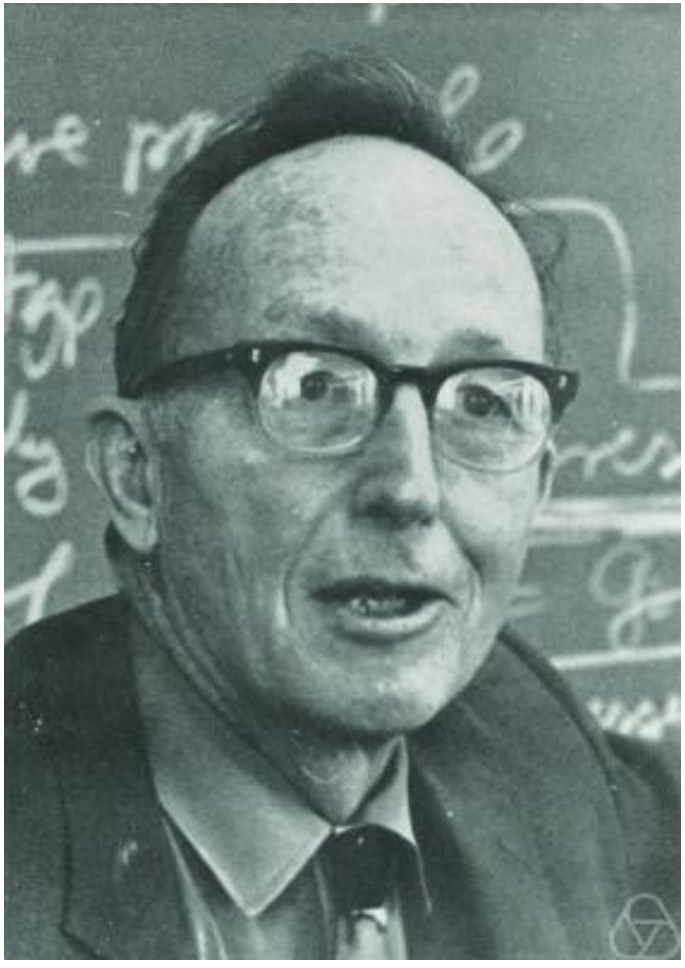
Dopo

Logica matematica (Jon Barwise, 1977):

- teoria degli insiemi
- teoria della dimostrazione
- teoria dei modelli
- teoria della ricorsione

Oggi anche: teoria delle categorie, teoria dei topos, teoria dei tipi, ...

Dopo

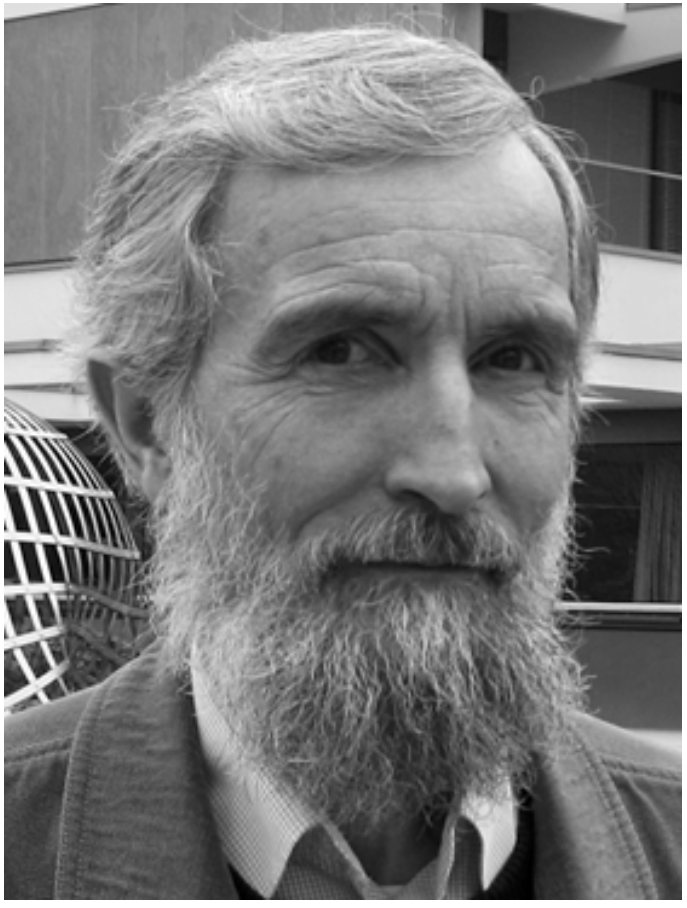


Saunders Mac Lane,
photo by Konrad Jacobs, 1972

Teoria delle categorie

Dopo

Teoria dei tipi



Per Martin-Löf

The Art of Ordinal Analysis

Michael Rathjen

Abstract. Ordinal analysis of theories is a core area of proof theory whose origins can be traced back to Hilbert's programme - the aim of which was to lay to rest all worries about the foundations of mathematics once and for all by securing mathematics via an absolute proof of consistency. Ordinal-theoretic proof theory came into existence in 1936, springing forth from Gentzen's head in the course of his consistency proof of arithmetic. The central theme of ordinal analysis is the classification of theories by means of transfinite ordinals that measure their 'consistency strength' and 'computational power'. The so-called *proof-theoretic ordinal* of a theory also serves to characterize its provably recursive functions and can yield both conservation and combinatorial independence results.

This paper intends to survey the development of "ordinally informative" proof theory from the work of Gentzen up to more recent advances in determining the proof-theoretic ordinals of strong subsystems of second order arithmetic.

Mathematics Subject Classification (2000). Primary 03F15, 03F05, 03F35; Secondary 03F03, 03-03.

Keywords. Proof theory, ordinal analysis, ordinal representation systems, proof-theoretic strength.

Riferimenti

Due testi classici sulla storia della matematica sono: *Carl B. Boyer*, A History of Mathematics, John Wiley & Sons (1968), e *Morris Kline*, Mathematical Thought from Ancient to Modern Times, Oxford University Press (1972). Anche, ottimi riferimenti sugli autori e sulle idee si possono trovare sulla Stanford Encyclopedia of Philosophy: `https://plato.stanford.edu/`

Per quelli interessati alla storia recente della logica, un libro breve e ben fatto è *Piergiorgio Odifreddi*, La matematica del Novecento—Dagli insiemi alla complessità, Piccola Biblioteca Einaudi, Einaudi, (2000).

Ci sono molti testi introduttivi alla logica matematica e alcune importanti guide di riferimento, tra cui menzioniamo *Jon Barwise*, Handbook of Mathematical Logic, Studies in Logic and the Foundations of Mathematics 90, North-Holland, (1977).

Sebbene non richiesto, gran parte del corso è basato su *John Bell* e *Moshe Machover* A Course in Mathematical Logic, North Holland (1977).

Logica

Lezione 2

Dr Marco Benini

marco.benini@uninsubria.it

Dipartimento di Scienza e Alta Tecnologia
Università degli Studi dell'Insubria

a.a. 2020/21

Syllabus

Logica proposizionale:

- Induzione
- Linguaggio
- Interpretazione intesa
- Sistema di deduzione

Dimostrazione per induzione

Intuitivamente, per dimostrare che una proprietà $P(x)$ vale per ogni possibile valore x in un dominio D, si potrebbe sostituire x con ogni valore $v \in D$ e dimostrare $P(v)$.
In generale, questo è impraticabile, e diventa impossibile quando D è infinito.

Tuttavia, se D può essere generato da un processo che ha una descrizione finita, o, in gergo, se D è *finitamente generato*, allora si può usare il processo di generazione per dimostrare $P(x)$.

Questa idea si chiama *induzione*.

Dimostrazione per induzione

Sia $\mathbb{N}$ l'insieme dei numeri naturali $\{0, 1, 2, \ldots\}$.

$\mathbb{N}$ può essere finitamente generato:

1. $0 \in \mathbb{N}$;
2. se $m \in \mathbb{N}$ allora $S(m) \in \mathbb{N}$, con S la funzione *successore*, $S(x) = x + 1$.

Quindi, se P è una proprietà, abbiamo un *principio di induzione*:

1. se abbiamo una prova di $P(0)$,
2. se siamo in grado di dimostrare $P(S(m))$ dall'ipotesi $P(m)$,

allora $P(x)$ vale per ogni $x \in \mathbb{N}$.

Dimostrazione per induzione

Il principio di induzione è giustificato intuitivamente:

- abbiamo una dimostrazione π_0 di $P(0)$ per 1;
- componendo π_0 con la prova dal punto 2 precedente, otteniamo una dimostrazione π_1 di $P(1)$;
- componendo π_1 con la prova dal punto 2 precedente, otteniamo una dimostrazione π_2 di $P(2)$;
- e così via ...

Perciò, per qualunque valore $v \in \mathbb{N}$, possiamo trovare una dimostrazione di $P(v)$ nella lista sopra.

Esempio

Enunciato 2.1

$$\sum_{n=0}^{k} n = \frac{k(k+1)}{2} \quad .$$

Dimostrazione.

Per induzione su $k \in \mathbb{N}$:

1. quando $k = 0$, $\sum_{n=0}^{0} n = 0 = \frac{0(0+1)}{2}$;
2. assumiamo $\sum_{n=0}^{k} n = \frac{k(k+1)}{2}$. Quindi,

$$\sum_{n=0}^{k+1} n = k+1+\sum_{n=0}^{k} n = k+1+\frac{k(k+1)}{2} = \frac{k^2+3k+2}{2} = \frac{(k+1)((k+1)+1)}{2} \quad .$$

□

Definizione per induzione

L'induzione può essere usata per definire nuovi concetti e nuovi oggetti.

Sia τ una mappa da $\mathbb{N}$. Quindi, l'immagine di τ è un nuovo concetto, e i suoi elementi sono nuovi oggetti.

Per esempio, ponendo $\tau(x) = 2x$, definiamo un nuovo concetto: i *numeri pari*.

Considerando la collezione di tutte le mappe $\rho : \mathbb{N} \to \{0, \ldots, 9\}$ all'insieme delle cifre, otteniamo nuovi oggetti, uno per ogni ρ, che denotano i numeri reali $0.\rho(0)\rho(1)\cdots$, la cui cifra alla posizione decimale n è $\rho(n)$, e un nuovo concetto, l'*intervallo unitario* $[0,1]$.

In generale

L'idea di induzione è molto più generale: ogni volta che possiamo generare finitamente un dominio, abbiamo un principio di induzione che può essere usato per ragionare e per definire.

In Logica Matematica, l'induzione è uno degli strumenti fondamentali e dei più potenti.

Logica proposizionale

Logica proposizionale

L'idea della logica proposizionale è che una *proposizione* è un oggetto che rappresenta un *valore di verità*, che sia *vero* oppure *falso*. Proposizioni composte ottengono il loro valore di verità dalle componenti, mentre le proposizioni di base hanno un valore di verità che dipende dal mondo in cui vengono interpretate.

Per esempio, la frase "Socrate è un uomo" può essere vera o falsa, dato che Socrate può riferirsi all'antico filosofo greco, oppure ad un gatto. D'altro canto, "Se Socrate è un uomo allora Socrate è mortale" è vera quando Socrate è sia un uomo che mortale, ma è anche vera quando Socrate non è un uomo, ed è falsa quando Socrate è un uomo immortale.

Linguaggio

Definizione 2.2 (Formula)

Sia $\mathscr{V}$ un insieme infinito di simboli, dette *variabili*, che non contenga '(', ')', '$\top$', '$\bot$', '$\wedge$', '$\vee$', '$\supset$', '$\neg$'.
Allora, una *formula* è induttivamente definita come

1. una variabile $x \in \mathscr{V}$ è una formula;
2. $\top$, letto come *vero*, e $\bot$, *falso*, sono formule;
3. se A è una formula, anche $(\neg A)$, *non*, *negazione*, lo è;
4. se A e B sono formule, anche $(A \wedge B)$, *e*, *congiunzione*, $(A \vee B)$, *o*, *disgiunzione*, e $(A \supset B)$, *implicazione*, lo sono.

Si noti come A e B sopra non sono parte del linguaggio, ma sono variabili nel metalinguaggio—noi saremo piuttosto informali rispetto al metalinguaggio, ovvero il linguaggio usato per descrivere il linguaggio logico.

Linguaggio

Per semplificare la notazione, adottiamo un insieme di abbreviazioni:

- le parentesi più esterne non vengono scritte: $x \wedge y$ invece di $(x \wedge y)$;
- la congiunzione e la disgiunzione hanno una precedenza maggiore rispetto all'implicazione: $x \wedge y \supset z \vee w$ invece di $((x \wedge y) \supset (z \vee w))$;
- la negazione ha una precedenza maggiore rispetto alla congiunzione, alla disgiunzione e all'implicazione: $\neg x \wedge \neg y$ invece di $((\neg x) \wedge (\neg y))$;
- le lettere minuscole, quando non detto diversamente, stanno per variabili.
- le lettere maiuscole, quando non specificato diversamente, stanno per oggetti nel metalinguaggio.

Un punto importante da sottolineare è che la definizione di formula è per induzione. Quindi possiamo usare questa struttura per definire nuove nozioni o per dimostrare proprietà delle formule.

Linguaggio

Come esempio di definizione induttiva, definiamo la nozione di *sottoformula*:

Definizione 2.3 (Sottoformula)

Data una formula A sull'insieme V di variabili, B è una *sottoformula* di A se e solo se B appartiene all'insieme $S(A)$ induttivamente definito come

1. se $A \in V$, $A \equiv \top$, o $A \equiv \bot$ allora $S(A) = \{A\}$;
2. se $A \equiv B \wedge C$, $A \equiv B \vee C$, o $A \equiv B \supset C$ allora $S(A) = \{A\} \cup S(B) \cup S(C)$;
3. se $A \equiv \neg B$ allora $S(A) = \{A\} \cup S(B)$.

Diciamo in modo equivalente che B *occorre* in A, per intendere che B è una sottoformula di A.

In generale, il simbolo $\equiv$ nel metalinguaggio significa "letteralmente uguale", scritto esattamente nello stesso modo.

Interpretazione intesa

Informalmente, un *valore di verità* è vero oppure falso.

- Una variabile x denota un qualche valore di verità.
- $\top$ denota vero.
- $\bot$ denota falso.
- $A \wedge B$ è vera quando sia A che B sono vere, ed è falsa altrimenti.
- $A \vee B$ è vera quando A è vera oppure quando B è vera, oppure quando entrambe sono vere, ed è falsa quando sia A che B sono false.
- $A \supset B$ è vera se, quando A è vera, anche B è vera, ed è vera anche quando A è falsa. Essa è falsa quando A è vera ma B è falsa.
- $\neg A$ è vera esattamente quando A è falsa.

In generale, il valore di verità di una formula dipende dai valori delle proprie variabili. Talvolta, può accadere che una formula sia vera indipendentemente dal valore delle proprie variabili, ad esempio, $x \supset x$ è vera per qualunque valore di verità x possa assumere.

La logica è principalmente interessata a studiare le *tautologie*, quelle formule che sono vere indipendentemente dal valore delle proprie variabili.

Deduzione naturale

Un modo ovvio per scoprire se una formula sia vera consiste nel provare tutti i possibili valori per le variabili che vi occorrano.

Però ci sono tre principali difetti in questa strategia:

- la strategia è esponenziale: se ci sono n variabili distinte in una formula, dobbiamo provare 2^n assegnamenti possibili.
- la strategia non scala ad altri sistemi logici. Per esempio, prendiamo l'aritmetica: non è fattibile mostrare la verità di una formula provando tutti i possibili valori per le sue variabili poiché ognuna di esse denota un numero naturale!
- la strategia non fornisce alcuna spiegazione: non abbiamo idea del perché una formula valga, eccetto che essa soddisfa esaustivamente tutti i possibili assegnamenti. In particolare, non sappiamo quali assiomi nella teoria sono richiesti per rendere vera la proprietà.

Ciò che vogliamo è una nozione di *prova*: un modo di ragionare che, partendo da alcuni fatti di base accettati e adottando una serie di regole accettate, ci permetta di concludere che una formula sia vera.

Deduzione naturale

Definizione 2.4 (Teoria)

Fissato un linguaggio, una *teoria* T è un insieme di formule, ognuna normalmente denominata *assioma*.
Quando $T = \emptyset$, diremo che la teoria è la *logica pura*.

Definizione 2.5 (Prova)

Fissato un linguaggio e una teoria T in esso, una *prova* o *deduzione* o *dimostrazione* di una formula A, la *conclusione*, da un insieme Γ di formule, le *ipotesi* o *assunzioni*, è induttivamente definita da un insieme di regole di inferenza presentate nei lucidi che seguono.
Una formula A che sia la conclusione di una prova senza assunzioni, è detta *teorema* nella teoria T.

Deduzione naturale

Le regole di inferenza che governano le congiunzioni sono:

$$\frac{A \wedge B}{A}\,\wedge\mathrm{E}_1 \qquad \frac{A \wedge B}{B}\,\wedge\mathrm{E}_2 \qquad \frac{A \quad B}{A \wedge B}\,\wedge\mathrm{I}$$

abbiamo due regole di eliminazione e una regola di introduzione.

Le regole che governano la disgiunzione sono:

$$\frac{A}{A \vee B}\,\vee\mathrm{I}_1 \qquad \frac{B}{A \vee B}\,\vee\mathrm{I}_2 \qquad \frac{A \vee B \quad \begin{array}{c}[A]\\ \vdots \\ C\end{array} \quad \begin{array}{c}[B]\\ \vdots \\ C\end{array}}{C}\,\vee\mathrm{E}$$

Deduzione naturale

L'implicazione e la negazione sono soggette alle seguenti regole:

$$\dfrac{\begin{array}{c}[A]\\ \vdots \\ B\end{array}}{A \supset B}\supset\mathsf{I} \qquad \dfrac{A \supset B \quad A}{B}\supset\mathsf{E}$$

$$\dfrac{\begin{array}{c}[A]\\ \vdots \\ \bot\end{array}}{\neg A}\neg\mathsf{I} \qquad \dfrac{\neg A \quad A}{\bot}\neg\mathsf{E}$$

Esse sono molto simili poiché, come vedremo nella prossima lezione, la negazione può essere definita a partire dall'implicazione.

Deduzione naturale

Il vero e il falso sono governati dalle seguenti regole:

$$\frac{}{\top}\,\top\mathsf{I} \qquad \frac{\bot}{A}\,\bot\mathsf{E}$$

Se A è un assioma della teoria T, ovvero se $A \in T$, siamo autorizzati a dedurlo:

$$\frac{}{A}\,\mathsf{ax}$$

Se A è una assunzione, ovvero se $A \in \Gamma$, allora possiamo dedurla

$$A$$

Deduzione naturale

Infine, per ogni formula A, A è vera oppure è falsa. Questo fatto è espresso nella Legge del Terzo Escluso:

$$\frac{}{A \vee \neg A}\ \text{lem}$$

Come vedremo più avanti nel corso, la Legge del Terzo Escluso è *delicata*, e pertanto essa ha uno speciale status.

In generale, per quanto possibile, cercheremo di evitare il suo uso nelle dimostrazioni.

Deduzione naturale

Qualche commento:

- eccetto che per la Legge del Terzo Escluso, le regole vanno in coppia: ogni connettivo ha associato una o più regole di introduzione, e una o più regole di eliminazione.
- le assunzioni possono essere *libere* o *scaricate*. Le ipotesi libere sono reali, nel senso che la dimostrazione dipende da esse; le ipotesi scaricate sono usate per eliminare una assunzione locale, che non influisce sull'intera prova. Questo si capisce meglio guardando le regola di "introduzione di implicazione": per provare $A \supset B$, assumiamo localmente A e proviamo a dimostrare B, ma il risultato finale non dipende più da A.
- scaricare le ipotesi è opzionale: non possiamo scaricare una assunzione se la regola non lo permette, ma possiamo scaricare (oppure possiamo non farlo) una ipotesi se la regola lo permette.

Quando non vogliamo specificare i dettagli della prova, scriviamo $\pi : \Gamma \vdash_T A$, intendendo che π è una dimostrazione di A dalle ipotesi Γ nella teoria T. Quando la prova è irrilevante, omettiamo il π; quando la teoria è nota oppure vuota, omettiamo il T; quando l'insieme delle ipotesi è vuoto, omettiamo il Γ.

Quadro riassuntivo

$$\frac{A \quad B}{A \wedge B}\,\wedge\mathrm{I} \qquad \frac{A \wedge B}{A}\,\wedge\mathrm{E}_1 \qquad \frac{A \wedge B}{B}\,\wedge\mathrm{E}_2 \qquad \frac{\bot}{A}\,\bot\mathrm{E}$$

$$\frac{A}{A \vee B}\,\vee\mathrm{I}_1 \qquad \frac{B}{A \vee B}\,\vee\mathrm{I}_2 \qquad \frac{A \vee B \quad \begin{array}{c}[A]\\ \vdots\\ C\end{array} \quad \begin{array}{c}[B]\\ \vdots\\ C\end{array}}{C}\,\vee\mathrm{E} \qquad \frac{}{\top}\,\top\mathrm{I}$$

$$\frac{\begin{array}{c}[A]\\ \vdots\\ B\end{array}}{A \supset B}\,\supset\mathrm{I} \qquad \frac{A \supset B \quad A}{B}\,\supset\mathrm{E} \qquad \boxed{\frac{}{A \vee \neg A}\,\mathrm{lem}} \qquad \frac{\begin{array}{c}[A]\\ \vdots\\ \bot\end{array}}{\neg A}\,\neg\mathrm{I} \qquad \frac{\neg A \quad A}{\bot}\,\neg\mathrm{E}$$

Note finali

Questa lezione è fondamentale. Siete tenuti a memorizzare le regole di inferenza nel lucido precedente e ad usarle quando occorra.

Sebbene il significato inteso sembri ovvio, siate sicuri di aver davvero compreso il modo in cui interpretiamo l'implicazione.

Prendetevi qualche tempo per notare le simmetrie tra le regole d'inferenza:

- eccetto che per la Legge del Terzo Escluso, ci sono regole di introduzione e di eliminazione per ogni connettivo;
- non si può introdurre il falso;
- non si può eliminare il vero;
- l'implicazione e la negazione sono simili;
- la congiunzione e la disgiunzione sono simili.

Prendetevi il tempo di studiare gli esempi che seguono: ad un certo momento, ci si attenderà che voi siate in grado di svolgere da soli dimostrazioni come quelle presentate.

Riferimenti

La deduzione naturale, nel formato in cui è stata presentata, è stata introdotta nel testo classiso *Dag Prawitz*, Natural Deduction, Almqvist & Wiksell, Stockholm, (1965). Recentemente, questo testo è stato ristampato da Dover.

Noi useremo principalmente il testo *John Bell* e *Moshé Machover*, A Course in Mathematical Logic, North-Holland, (1977) come riferimento generale in questo corso. Sebbene sia un libro datato, è ancora oggi un riferimento classico, e contiene un trattamento completo e formale di tutte le nozioni.

Per una esposizione vasta e profonda della deduzione naturale si veda *Anne S. Troelstra* e *Helmut Schwichtenberg*, Basic Proof Theory, Cambridge Tracts in Theoretical Computer Science 43, Cambridge University Press, (1996). Questo libro si estende ben oltre il contenuto di questo corso.

Logica

Lezione 3

Dr Marco Benini

marco.benini@uninsubria.it

Dipartimento di Scienza e Alta Tecnologia
Università degli Studi dell'Insubria

a.a. 2020/21

Syllabus

Logica proposizionale:

- Esempi
- Tecniche di dimostrazione

Esempi

Per dimostrare una formula, abbiamo bisogno di pensare al contrario, a partire dalla conclusione: quindi le regole di introduzione eliminano il connettivo principale da una formula.

La prima tecnica di base è ridurre una formula da dimostrare applicando l'unica regola di introduzione che possa generarla.

Esempio 3.1

Si dimostri $\vdash A \supset (B \supset A)$

$$\dfrac{\dfrac{[A]^1}{B \supset A}\supset \mathsf{I}}{A \supset (B \supset A)}\supset \mathsf{I}^1$$

Esempi

Un utile ausilio nel dimostrare una formula è tenere traccia delle ipotesi che generiamo nel processo di prova.

Nell'ultimo esempio, siamo partiti da

$$A \supset (B \supset A)$$

Abbiamo provato a semplificare l'obiettivo da dimostrare per mezzo della regola di introduzione di implicazione

$$\frac{B \supset A}{A \supset (B \supset A)} \supset \mathrm{I}$$

e, nel contempo, il nostro insieme di ipotesi, che era inizialmente vuoto, è divenuto $\{A\}$.

Quindi, abbiamo provato a semplificare l'obiettivo corrente $B \supset A$, ottenendo

$$\frac{\dfrac{A}{B \supset A} \supset \mathrm{I}}{A \supset (B \supset A)} \supset \mathrm{I}$$

e, nel frattempo, il nostro insieme di assunzioni è diventato $\{A, B\}$.

Esempi

E, adesso, vediamo che l'obiettivo da dimostrare è nell'insieme delle ipotesi, quindi possiamo concludere la dimostrazione scaricando l'ipotesi.

$$\dfrac{\dfrac{[A]^1}{B \supset A}\supset \mathsf{I}}{A \supset (B \supset A)}\supset \mathsf{I}^1$$

Vale la pena notare che

- dobbiamo ricordarci quale regola abbia introdotto quale assunzione, in modo tale che lo scaricamento possa essere effettuato correttamente;
- è possibile che vi siano ipotesi inutilizzate, come B nell'esempio.

Esempi

Quando una assunzione è una formula complessa, vale la pena smontarla attraverso una regola di eliminazione.

Esempio 3.2

Si dimostri $\vdash (A \supset C) \supset ((B \supset C) \supset (A \vee B \supset C))$

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{[A \vee B]^1 \quad \dfrac{[A]^2 \quad [A \supset C]^3}{C}\supset E \quad \dfrac{[B]^2 \quad [B \supset C]^4}{C}\supset E}{C}\vee E^2}{A \vee B \supset C}\supset I^1}{(B \supset C) \supset (A \vee B \supset C)}\supset I^4}{(A \supset C) \supset ((B \supset C) \supset (A \vee B \supset C))}\supset I^3$$

Si noti come le assunzioni sono locali ad una sottoprova.

Esempi

Esempio 3.3

Si provi che $\vdash A \vee B = B \vee A$ ($\vee$ è commutativo).

Notiamo che la proprietà è auto-duale

$$\dfrac{\dfrac{[A \vee B]^1 \quad \dfrac{[A]^2}{B \vee A} \vee I_2 \quad \dfrac{[B]^2}{B \vee A} \vee I_1}{B \vee A} \vee E^2}{A \vee B \supset B \vee A} \supset I^1$$

Esempi

Esempio 3.4

Si provi $\vdash A \wedge B = B \wedge A$ ($\wedge$ è commutativo).

$$\dfrac{\dfrac{\dfrac{[A \wedge B]^1}{B}\wedge E_2 \quad \dfrac{[A \wedge B]^1}{A}\wedge E_1}{B \wedge A}\wedge I}{A \wedge B \supset B \wedge A}\supset I^1$$

Esempi

Ci può essere più di un modo per dimostrare un risultato.

Esempio 3.5

Si dimostri $\vdash A \vee A = A$ ($\vee$ è idempotente).

$$\dfrac{\dfrac{[A \vee A]^1 \quad [A]^2 \quad [A]^2}{A}\,\vee E^2}{A \vee A \supset A}\,\supset I^1 \qquad \dfrac{\dfrac{[A]^1}{A \vee A}\,\vee I_1}{A \supset A \vee A}\,\supset I^1 \qquad \dfrac{\dfrac{[A]^1}{A \vee A}\,\vee I_2}{A \supset A \vee A}\,\supset I^1$$

Esempi

Esempio 3.6

Si provi $\vdash A \wedge A = A$ ($\wedge$ è idempotente).

$$\dfrac{\dfrac{[A \wedge A]^1}{A}\,\wedge E_1}{A \wedge A \supset A}\,\supset I^1 \qquad \dfrac{\dfrac{[A \wedge A]^1}{A}\,\wedge E_2}{A \wedge A \supset A}\,\supset I^1 \qquad \dfrac{\dfrac{[A]^1 \quad [A]^1}{A \wedge A}\,\wedge I}{A \supset A \wedge A}\,\supset I^1$$

Esempi

Esempio 3.7

Si provi $\vdash A \vee (A \wedge B) = A$ (legge di assorbimento).

$$\dfrac{\dfrac{[A \vee (A \wedge B)]^1 \quad [A]^2 \quad \dfrac{[A \wedge B]^2}{A} \wedge E_1}{A} \vee E^2}{A \vee (A \wedge B) \supset A} \supset I^1 \qquad \dfrac{\dfrac{[A]^1}{A \vee (A \wedge B)} \vee I_1}{A \supset A \vee (A \wedge B)} \supset I^1$$

Esempi

Esempio 3.8

Si dimostri $\vdash A \wedge (A \vee B) = A$ (legge di assorbimento).

$$\dfrac{\dfrac{[A \wedge (A \vee B)]^1}{A}\,\wedge E_1}{A \wedge (A \vee B) \supset A}\,\supset I^1 \qquad \dfrac{\dfrac{[A]^1 \quad \dfrac{[A]^1}{A \vee B}\,\vee I_1}{A \wedge (A \vee B)}\,\wedge I}{A \supset A \wedge (A \vee B)}\,\supset I^1$$

Esempi

L'eliminazione del falso permette di dedurre qualsiasi formula uno desideri. Ma il falso deriva sempre da una contraddizione.

Esempio 3.9

Si dimostri $\vdash \neg A \supset (A \supset B)$.

$$\dfrac{\dfrac{\dfrac{\dfrac{[\neg A]^1 \quad [A]^2}{\bot}\neg\mathsf{E}}{B}\bot\mathsf{E}}{A \supset B}\supset\mathsf{I}^2}{\neg A \supset (A \supset B)}\supset\mathsf{I}^1$$

Esempi

Pensando all'indietro, l'introduzione della negazione permette di assumere la conclusione deprivata della negazione. È una forma di ragionamento per contraddizione.

Esempio 3.10

Si provi $\vdash A \wedge B \supset \neg(A \supset \neg B)$.

$$\dfrac{\dfrac{\dfrac{[A \supset \neg B]^1 \qquad \dfrac{[A \wedge B]^2}{A}\,\wedge E_1}{\neg B}\,\supset E \qquad \dfrac{[A \wedge B]^2}{B}\,\wedge E_2}{\bot}\,\neg E}{\neg(A \supset \neg B)}\,\neg I^1}{A \wedge B \supset \neg(A \supset \neg B)}\,\supset I^2$$

Esempi

Esempio 3.11

Si dimostri $\vdash \neg(A \vee B) = \neg A \wedge \neg B$ (Legge di De Morgan).

$$\dfrac{\dfrac{\dfrac{[\neg(A \vee B)]^1 \quad \dfrac{[A]^2}{A \vee B}\vee I_1}{\bot}\neg E}{\neg A}\neg I^2 \quad \dfrac{\dfrac{[\neg(A \vee B)]^1 \quad \dfrac{[B]^3}{A \vee B}\vee I_2}{\bot}\neg E}{\neg B}\neg I^3}{\dfrac{\neg A \wedge \neg B}{\neg(A \vee B) \supset \neg A \wedge \neg B}\supset I^1}\wedge I$$

$$\dfrac{[A \vee B]^1 \quad \dfrac{[A]^2 \quad \dfrac{[\neg A \wedge \neg B]^3}{\neg A}\wedge E_1}{\bot}\neg E \quad \dfrac{[B]^2 \quad \dfrac{[\neg A \wedge \neg B]^3}{\neg B}\wedge E_2}{\bot}\neg E}{\dfrac{\dfrac{\bot}{\neg(A \vee B)}\neg I^1}{\neg A \wedge \neg B \supset \neg(A \vee B)}\supset I^3}\vee E^2$$

Esempi

Esempio 3.12

Si provi $\vdash \neg\neg(A \wedge B) \supset \neg\neg A \wedge \neg\neg B$.

$$\dfrac{\dfrac{\dfrac{\dfrac{[\neg\neg(A\wedge B)]^{1} \qquad \dfrac{\dfrac{[\neg A]^{2} \qquad \dfrac{[A\wedge B]^{3}}{A}\wedge E_1}{\bot}\neg E}{\neg(A\wedge B)}\neg I^{3}}{\bot}\neg E}{\neg\neg A}\neg I^{2} \qquad \dfrac{\dfrac{\dfrac{[\neg\neg(A\wedge B)]^{1} \qquad \dfrac{\dfrac{[\neg B]^{4} \qquad \dfrac{[A\wedge B]^{5}}{B}\wedge E_2}{\bot}\neg E}{\neg(A\wedge B)}\neg I^{5}}{\bot}\neg E}{\neg\neg B}\neg I^{4}}{\neg\neg A\wedge\neg\neg B}\wedge I}{\neg\neg(A\wedge B)\supset\neg\neg A\wedge\neg\neg B}\supset I^{1}$$

Esempi

Esempio 3.13

Si provi $\vdash \neg\neg A \wedge \neg\neg B \supset \neg\neg(A \wedge B)$.

$$\dfrac{\dfrac{\dfrac{\dfrac{[A]^1 \quad [B]^2}{A \wedge B}\wedge\mathrm{I} \qquad [\neg(A \wedge B)]^3}{\bot}\neg\mathrm{E}}{\neg A}\neg\mathrm{I}^1 \qquad \dfrac{[\neg\neg A \wedge \neg\neg B]^4}{\neg\neg A}\wedge\mathrm{E}_1}{\bot}\neg\mathrm{E} \qquad \dfrac{[\neg\neg A \wedge \neg\neg B]^4}{\neg\neg B}\wedge\mathrm{E}_2}{\bot}\neg\mathrm{E}}{\neg B}\neg\mathrm{I}^2}{\neg\neg(A \wedge B)}\neg\mathrm{I}^3}{\neg\neg A \wedge \neg\neg B \supset \neg\neg(A \wedge B)}\supset\mathrm{I}^4$$

Esempi

Esempio 3.14

Dimostrare $\vdash \neg\neg(A \supset B) \supset \neg\neg A \supset \neg\neg B$.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{[A]^1 \quad [A \supset B]^2}{B}\supset E \quad [\neg B]^3}{\bot}\neg E}{\neg(A \supset B)}\neg I^2 \quad [\neg\neg(A \supset B)]^4}{\bot}\neg E}{\neg A}\neg I^1 \quad [\neg\neg A]^5}{\bot}\neg E}{\neg\neg B}\neg I^3}{\neg\neg A \supset \neg\neg B}\supset I^5}{\neg\neg(A \supset B) \supset \neg\neg A \supset \neg\neg B}\supset I^4$$

Esempi

Esempio 3.15

Provare $\vdash (\neg\neg A \supset \neg\neg B) \supset \neg\neg(A \supset B)$.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{[\neg\neg A \supset \neg\neg B]^{2} \qquad \dfrac{\dfrac{[A]^{3} \quad [\neg A]^{4}}{\bot}\,\neg E}{\neg\neg A}\,\neg I^{4}}{\neg\neg B}\,\supset E \qquad \dfrac{\dfrac{[\neg(A \supset B)]^{1} \quad \dfrac{[B]^{5}}{A \supset B}\,\supset I}{\bot}\,\neg E}{\neg B}\,\neg I^{5}}{\bot}\,\neg E}{B}\,\bot E}{A \supset B}\,\supset I^{3} \qquad [\neg(A \supset B)]^{1}}{\bot}\,\neg E}{\neg\neg(A \supset B)}\,\neg I^{1}}{(\neg\neg A \supset \neg\neg B) \supset \neg\neg(A \supset B)}\,\supset I^{2}$$

Esempi

Le dimostrazioni che richiedano la Legge del Terzo Escluso sono più difficili. La strategia fondamentale è che una applicazione di quel principio è richiesta quando nessun'altra strategia sia possibile.

Esempio 3.16

Provare $\vdash A = \neg\neg A$ (legge di doppia negazione).

$$\dfrac{\dfrac{\dfrac{}{A \vee \neg A}\,\text{lem} \qquad \dfrac{\dfrac{[\neg A]^1 \quad [\neg\neg A]^2}{\bot}\,\neg\text{E}}{A}\,\bot\text{E} \qquad [A]^1}{A}\,\vee\text{E}^1}{\neg\neg A \supset A}\,\supset\text{I}^2$$

$$\dfrac{\dfrac{\dfrac{[\neg A]^1 \quad [A]^2}{\bot}\,\neg\text{E}}{\neg\neg A}\,\neg\text{I}^1}{A \supset \neg\neg A}\,\supset\text{I}^2$$

Esempi

Non dobbiamo affidarci alla forma del teorema! Varianti anche piccole nella forma possono essere dimostrabili **senza** ricorrere al terzo escluso.

Esempio 3.17

Dimostrare $\vdash \neg A = \neg\neg\neg A$.

$$\dfrac{\dfrac{[\neg\neg\neg A]^1 \qquad \dfrac{\dfrac{[\neg A]^2 \quad [A]^3}{\bot}\neg E}{\neg\neg A}\neg I^2}{\neg A}\neg E}{\neg\neg\neg A \supset \neg A}\supset I^1 \qquad \dfrac{\dfrac{[\neg\neg A]^1 \quad [\neg A]^2}{\bot}\neg E}{\neg\neg\neg A}\neg I^1}{\neg A \supset \neg\neg\neg A}\supset I^2$$

Esempi

Si potrebbe pensare che la Legge del Terzo Escluso riguardi la negazione. Questo è sbagliato: ci sono fatti elementari, in cui la negazione non compare, che **hanno bisogno** del terzo escluso per essere dimostrati.

Esempio 3.18

Si dimostri $\vdash (A \supset B) \vee (B \supset A)$.

$$\dfrac{\dfrac{}{A \vee \neg A}\,\text{lem} \quad \dfrac{\dfrac{[A]^1}{B \supset A}\,\supset\text{I}}{(A \supset B) \vee (B \supset A)}\,\vee\text{I}_2 \quad \dfrac{\dfrac{\dfrac{[A]^2 \quad [\neg A]^1}{\bot}\,\neg\text{E}}{B}\,\bot\text{E}}{A \supset B}\,\supset\text{I}^2}{(A \supset B) \vee (B \supset A)}\,\vee\text{I}_1 \qquad \vee\text{E}^1$$

Esempi

Esempio 3.19

Provare $\vdash ((A \supset B) \supset A) \supset A$ (legge di Pierce).

$$\dfrac{\dfrac{\dfrac{}{A \vee \neg A}\,\text{lem} \quad \dfrac{[A]^1}{\dfrac{\dfrac{\dfrac{[\neg A]^1 \quad [A]^3}{\bot}\,\neg\text{E}}{B}\,\bot\text{E}}{A \supset B}\,\supset\text{I}^3 \qquad [(A \supset B) \supset A]^2}{A}\,\supset\text{E}}{A}\,\vee\text{E}^1}{((A \supset B) \supset A) \supset A}\,\supset\text{I}^2$$

Esempi

Esempio 3.20

Si dimostri $\vdash A \supset B = \neg B \supset \neg A$ (contronominale).

$$\dfrac{\dfrac{\dfrac{[A \supset B]^1 \quad [A]^2}{B}\supset E \qquad [\neg B]^3}{\bot}\neg E}{\dfrac{\neg A}{\neg B \supset \neg A}\supset I^3}\neg I^2 \Big/ (A \supset B) \supset (\neg B \supset \neg A)\ \supset I^1$$

$$\dfrac{\dfrac{\dfrac{\dfrac{[\neg B \supset \neg A]^2 \quad [\neg B]^1}{\neg A}\supset E \qquad [A]^3}{\bot}\neg E}{B}\bot E \qquad \dfrac{}{B \vee \neg B}\text{lem} \quad [B]^1}{\dfrac{B}{A \supset B}\supset I^3}\vee E^1 \Big/ (\neg B \supset \neg A) \supset (A \supset B)\ \supset I^2$$

Esempi

Esempio 3.21

Provare $\vdash A \supset B = \neg(A \wedge \neg B)$.

$$\dfrac{\dfrac{\dfrac{[A \supset B]^1 \quad \dfrac{[A \wedge \neg B]^2}{A}\wedge E_1}{B}\supset E \quad \dfrac{[A \wedge \neg B]^2}{\neg B}\wedge E_2}{\bot}\neg E}{\dfrac{\dfrac{\bot}{\neg(A \wedge \neg B)}\neg I^2}{(A \supset B) \supset \neg(A \wedge \neg B)}\supset I^1}$$

$$\dfrac{\dfrac{\dfrac{\dfrac{}{B \vee \neg B}\,\text{lem} \quad [B]^1 \quad \dfrac{\dfrac{[A]^2 \quad [\neg B]^1}{A \wedge \neg B}\wedge I \quad [\neg(A \wedge \neg B)]^3}{\bot}\neg E}{B}\vee E^1}{A \supset B}\supset I^2}{\neg(A \wedge \neg B) \supset (A \supset B)}\supset I^3$$

Esempio 3.22

Dimostrare $\vdash A \vee B = \neg A \supset B$.

$$
\dfrac{\dfrac{[A \vee B]^1 \qquad \dfrac{\dfrac{[A]^2 \quad [\neg A]^3}{\bot}\,\neg\mathrm{E}}{B}\,\bot\mathrm{E} \qquad [B]^2}{\dfrac{B}{\neg A \supset B}\,\supset\mathrm{I}^3}\,\vee\mathrm{E}^2}{A \vee B \supset (\neg A \supset B)}\,\supset\mathrm{I}^1
$$

$$
\dfrac{\dfrac{\dfrac{}{A \vee \neg A}\,\mathrm{lem} \qquad \dfrac{[A]^1}{A \vee B}\,\vee\mathrm{I}_1 \qquad \dfrac{\dfrac{[\neg A]^1 \quad [\neg A \supset B]^2}{B}\,\supset\mathrm{E}}{A \vee B}\,\vee\mathrm{I}_2}{A \vee B}\,\vee\mathrm{E}^1}{(\neg A \supset B) \supset A \vee B}\,\supset\mathrm{I}^2
$$

Riferimenti

Esercizi possono essere trovati in ogni testo standard, ad esempio nel Capitolo 1 di *John Bell* e *Moshé Machover*, A Course in Mathematical Logic, North-Holland, (1977).

Alcuni esercizi aggiuntivi sono disponibili nella pagina web del corso.

Le tecniche di dimostrazione provengono dal teorema di completezza e dalla prova di normalizzazione.

Logica

Lezione 4

Dr Marco Benini

marco.benini@uninsubria.it

Dipartimento di Scienza e Alta Tecnologia
Università degli Studi dell'Insubria

a.a. 2020/21

Syllabus

Logica proposizionale:

- Semantica: tavole di verità
- Insiemi minimali di connettivi

Semantica

Il significato inteso della logica proposizionale può essere formalizzato. In questo modo otteniamo una prima semantica molto semplice per la sintassi introdotta nella seconda lezione.

Definizione 4.1 (Semantica a tavole di verità)

Fissata una mappa $\nu\colon V \to \{0,1\}$ dall'insieme delle variabili V all'insieme dei valori di verità, denotati da 0 e 1, il *significato* $[\![A]\!]$ di una formula A è induttivamente definito come segue:

- se $A \in V$ è una variabile allora $[\![A]\!] = \nu(A)$;
- $[\![\top]\!] = 1$;
- $[\![\bot]\!] = 0$;

↪

Semantica

↪ (Semantica a tavole di verità)

- se $A \equiv B \wedge C$ allora $[\![A]\!]$ è calcolato dalla funzione tabulata come

$[\![B]\!]$	$[\![C]\!]$	$[\![B \wedge C]\!]$
0	0	0
0	1	0
1	0	0
1	1	1

- se $A \equiv B \vee C$ allora $[\![A]\!]$ è calcolato dalla funzione tabulata come

$[\![B]\!]$	$[\![C]\!]$	$[\![B \vee C]\!]$
0	0	0
0	1	1
1	0	1
1	1	1

↪

Semantica

↪ (Semantica a tavole di verità)

- se $A \equiv \neg B$ allora $[\![A]\!]$ è calcolato dalla funzione tabulata come

$[\![B]\!]$	$[\![\neg B]\!]$
0	1
1	0

- se $A \equiv B \supset C$ allora $[\![A]\!]$ è calcolato dalla funzione tabulata come

$[\![B]\!]$	$[\![C]\!]$	$[\![B \supset C]\!]$
0	0	1
0	1	1
1	0	0
1	1	1

Esempi

Esempio 4.2

Mostrare che la formula $x \wedge y \supset x \vee y$ è vera indipendentemente dal valore che possiamo assegnare a x e y;

$[\![x]\!]$	$[\![y]\!]$	$[\![x \wedge y]\!]$	$[\![x \vee y]\!]$	$[\![x \wedge y \supset x \vee y]\!]$
0	0	0	0	1
0	1	0	1	1
1	0	0	1	1
1	1	1	1	1

Le corrispondenti derivazioni in calcolo naturale sono:

$$\dfrac{\dfrac{\dfrac{[x \wedge y]^1}{x}\wedge E_1}{x \vee y}\vee I_1}{x \wedge y \supset x \vee y}\supset I^1 \qquad \dfrac{\dfrac{\dfrac{[x \wedge y]^1}{y}\wedge E_2}{x \vee y}\vee I_2}{x \wedge y \supset x \vee y}\supset I^1$$

Esempi

Esempio 4.3

Le tavole di verità consentono di derivare anche proprietà espresse nella semantica: ad esempio che l'interpretazione di "e" è minore o uguale all'interpretazione di "o".

$[\![x]\!]$	$[\![y]\!]$	$[\![x \wedge y]\!]$	$[\![x \vee y]\!]$	$[\![x \wedge y]\!] \leq [\![x \vee y]\!]$
0	0	0	0	1
0	1	0	1	1
1	0	0	1	1
1	1	1	1	1

Applicazioni

Le tavole di verità sono ampiamente impiegate nella sintesi di circuiti logici, e vi sono molte tecniche di minimizzazione del numero di porte (connettivi). Queste tecniche sono basate sulle proprietà logiche dei connettivi.

Nell'ambito della logica, le tavole di verità non sono un modo efficiente per verificare se una formula è vera per ogni assegnamento delle sue variabili.

Infatti, il numero di assegnamenti possibili per una formula con n variabili è pari a 2^n, quindi sarebbe necessario provare un numero esponenziale di assegnamenti rispetto alla dimensione in numero di variabili della formula.

Interdipendenza dei connettivi

L'insieme dei connettivi è ridondante.

Enunciato 4.4

La negazione può essere definita dall'implicazione e il falso.

Dimostrazione.

Infatti $\neg A$ equivale a $A \supset \bot$:

A	$\neg A$	$A \supset \bot$
0	1	1
1	0	0

□

Interdipendenza dei connettivi

Enunciato 4.5

L'insieme dei connettivi $\wedge$, $\vee$ e $\neg$ basta per definire tutti gli altri.

Dimostrazione.

Per semplice verifica con le tavole di verità, si ottiene

- $\top$ può essere definito come $\neg X \vee X$, per una qualsiasi scelta di X;
- $\bot$ può essere definito come $\neg \top$;
- $A \supset B$ può essere definito come $\neg A \vee B$. $\square$

Interdipendenza dei connettivi

Enunciato 4.6

La congiunzione può essere definita dalla disgiunzione e dalla negazione. Anche, la disgiunzione può essere definita dalla congiunzione e dalla negazione.

Dimostrazione.

Con le tavole di verità è immediato verificare che

- $A \wedge B$ ha lo stesso valore di $\neg(\neg A \vee \neg B)$;
- $A \vee B$ ha lo stesso valore di $\neg(\neg A \wedge \neg B)$. □

Normalmente, $\neg(A \wedge B) = \neg A \vee \neg B$ e $\neg(A \vee B) = \neg A \wedge \neg B$ sono dette Leggi di De Morgan. Usiamo il simbolo $=$ tra due formule A e B intendendo che valgano sia $A \supset B$ che $B \supset A$, ovvero che A e B sono equivalenti.

Interdipendenza dei connettivi

Mettendo assieme i risultati precedenti, otteniamo che ognuno dei seguenti insiemi di connettivi è sufficiente per definire tutti gli altri:

- $\{\bot, \supset\}$;
- $\{\neg, \wedge\}$;
- $\{\neg, \vee\}$;
- $\{\neg, \supset\}$.

In principio, potremmo usare un singolo connettivo, sebbene questo non sia pratico nel ragionamento. Tuttavia, questo avviene, ad esempio, nella sintesi dei circuiti digitali. Definiamo $A \mid B = \neg(A \wedge B)$, noto come *segno di Sheffer* oppure **NAND**. Quindi, usando le tavole di verità, è semplice dimostrare

- $\neg A = A \mid A$;
- $A \supset B = A \mid (B \mid B)$.

Riferimenti

Le tavole di verità sono meglio descritte in un qualsiasi testo sull'architettura dei calcolatori, ad esempio, *Patterson Hennessy*, Computer organization and design — The hardware/software interface, Elsevier, 4ª edizione (2011). In tali testi sono analizzate anche le tecniche di minimizzazione dei circuiti e l'interdipendenza dei connettivi.

Logica

Lezione 5

Dr Marco Benini

marco.benini@uninsubria.it

Dipartimento di Scienza e Alta Tecnologia
Università degli Studi dell'Insubria

a.a. 2020/21

Syllabus

Logica proposizionale:

- Forma normale congiuntiva
- Forma normale disgiuntiva
- Soddisfacibilità

Forma normale

Quanto intendiamo mostrare è che ogni formula della logica proposizionale può essere espressa in modo equivalente mediante una formula, detta *forma normale*, che abbia una struttura prefissata.

Per *equivalente* intendiamo che, per ogni interpretazione delle variabili, il valore della formula coincida con il valore della sua forma normale.

Letterale

Definizione 5.1 (Letterale)

Fissato un linguaggio per la logica proposizionale, definiamo *letterale* una variabile o la negazione di una variabile.

Forma normale congiuntiva

Definizione 5.2 (Forma normale congiuntiva)

Una formula è detta essere in *forma normale congiuntiva* se essa è una congiunzione di disgiunzioni di letterali: $\bigwedge_i \bigvee_j l_{i,j}$.
Ognuna delle disgiunzioni è detta *clausola*.

Ad esempio:

- $(x \vee \neg y) \wedge (\neg x \vee z) \wedge (y \vee z)$ è una forma normale congiuntiva
- $(x \vee \neg y) \wedge (\neg x \vee z) \supset (y \vee z)$ non lo è — contiene una implicazione
- $(x \vee \neg y) \wedge (\neg x \vee z) \wedge (y \vee (z \wedge x))$ non lo è — $z \wedge x$ non è un letterale

Forma normale disgiuntiva

Definizione 5.3 (Forma normale disgiuntiva)

Una formula è detta essere in *forma normale disgiuntiva* se essa è una disgiunzione di congiunzioni di letterali: $\bigvee_i \bigwedge_j l_{i,j}$.
Ognuna delle congiunzioni è detta *implicante*.

Ad esempio: $(x \wedge \neg y) \vee (\neg x \wedge z) \vee (y \wedge z)$ è una forma normale disgiuntiva.

Forme normali

Teorema 5.4

Ogni formula proposizionale è equivalente ad una forma normale congiuntiva e ad una forma normale disgiuntiva.

Dimostrazione. (i)

Sia F una formula. Innanzitutto notiamo che possiamo eliminare i connettivi $=$ e $\supset$ usando le seguenti equivalenze:

- $A = B$ equivale a $(A \supset B) \wedge (B \supset A)$,
- $A \supset B$ equivale a $\neg A \vee B$,

Pertanto, non lede la generalità assumere che F contenga solo negazioni, congiunzioni e disgiunzioni. ↪

Forme normali

↪ Dimostrazione. (ii)

Procediamo per induzione sulla formula F:

- se F è una singola variabile, essa è in forma normale congiuntiva e disgiuntiva, banalmente.
- se $F = \neg F'$, per ipotesi induttiva F' equivale a $\bigwedge_i C_i$ in forma normale congiuntiva, e a $\bigvee_j D_j$ in forma normale disgiuntiva.
  Pertanto, F equivale a $\neg \bigwedge_i C_i$ e $\neg \bigvee_j D_j$.
  Applicando le leggi di De Morgan, $\neg(A \wedge B) = \neg A \vee \neg B$ e $\neg(A \vee B) = \neg A \wedge \neg B$, otteniamo che F equivale a $\bigvee_i \neg C_i$ e $\bigwedge_j \neg D_j$.
  Poiché $C_i = \bigvee_k l_{i,k}$ e $D_j = \bigwedge l_{j,k}$ con l un letterale, applicando nuovamente le leggi di De Morgan, F equivale a $\bigvee_i \bigwedge_k \neg l_{i,k}$ e $\bigwedge_j \bigvee_k \neg l_{j,k}$.
  Eliminando le eventuali doppie negazioni, queste due formule sono forme normali disgiuntive e congiuntive, rispettivamente. ↪

Forme normali

↪ Dimostrazione. (iii)

- Se $F = F_1 \vee F_2$, per ipotesi induttiva F_1 equivale a D_1 in forma normale disgiuntiva e C_1 in forma normale congiuntiva. Allo stesso modo per F_2 che equivale a D_2 e C_2.
  Pertanto F equivale a $D_1 \vee D_2$ che è una forma normale disgiuntiva.
  Consideriamo le clausole c_i^1 di C_1 e c_j^2 di C_2 e costruiamo $e_{i,j} = c_i^1 \vee c_j^2$.
  È immediato verificare che $\bigwedge_{i,j} e_{i,j}$ è in forma normale congiuntiva.
  Altrettanto semplice è verificare che $\bigwedge_{i,j} e_{i,j}$ equivale a F. ↪

Forme normali

↪ Dimostrazione. (iv)

- Il caso $F = F_1 \wedge F_2$ è completamente simmetrico. □

Si noti come la costruzione del Teorema 5.4 è algoritmica e fornisce una procedura effettiva per costruire la forma normale congiuntiva e disgiuntiva equivalente ad una formula data.

Complessità del calcolo delle forme normali

Esempio 5.5

Calcoliamo la forma normale disgiuntiva di

$$(x_1 \vee y_1) \wedge (x_2 \vee y_2) \wedge \cdots \wedge (x_n \vee y_n) \ .$$

Seguendo il procedimento del Teorema 5.4, otteniamo $\bigvee_i l_i$ dove gli l_i sono tutti gli implicanti $\bigwedge_{j=1}^{n} l_{i,j}$ con $l_{i,j} = x_j$ oppure $l_{i,j} = y_j$.

Notiamo come il numero di implicanti sia esponenziale rispetto al numero delle variabili, e questo è inevitabile.

Complessità del calcolo delle forme normali

La procedura del Teorema 5.4 permette di trasformare una formula F in una forma normale, congiuntiva o disgiuntiva.

Questa procedura è, nel caso peggiore, esponenziale in spazio (e quindi in tempo), come segue dall'esempio precedente e dalla struttura dell'algoritmo.

Soddisfacibilità

Definizione 5.6

Sia F una formula Booleana.

- F è una *tautologia* se F è vera **per ogni** assegnamento delle variabili.
- F è *soddisfacibile* se F è vera **per qualche** assegnamento delle variabili.
- F è *falsificabile* se F è falsa **per qualche** assegnamento delle variabili.
- F è *insoddisfacibile* se F è falsa **per ogni** assegnamento delle variabili.

Soddisfacibilità

Enunciato 5.7

Una formula F è insoddisfacibile se e solo se $\neg F$ è una tautologia.

Dimostrazione.

Se F è insoddisfacibile, allora è falsa per ogni assegnamento, quindi $\neg F$, per definizione di negazione, è vera per ogni assegnamento. Il ragionamento è reversibile. □

Soddisfacibilità

Definizione 5.8 (SAT)

Il problema *SAT* (per *SATisfiability*) è definito dalla coppia di insiemi $\langle \mathscr{F}, \mathscr{S} \rangle$ con $\mathscr{F}$ l'insieme di tutte le formule, e $\mathscr{S} = \{\phi : \phi \in \mathscr{F} \wedge \phi \text{ è soddisfacibile}\}$.

Questo è un *problema di decisione*: dato un elemento del primo insieme, si richiede un algoritmo per stabilire se l'elemento appartiene al secondo insieme.

Nel caso di SAT, è evidente che l'algoritmo che tenta tutti i possibili assegnamenti permette di risolvere il problema.

Nel caso di SAT, è anche evidente che questo algoritmo è esponenziale: su n variabili, è necessario testare 2^n assegnamenti.
Esiste un algoritmo polinomiale per risolvere SAT?

Soddisfacibilità

La risposta, sorprendente e difficile, è che *non lo sappiamo*!

Sappiamo che è molto improbabile che esista, ma non ne abbiamo la certezza assoluta. Tuttavia nessun informatico (o matematico) scommette sull'esistenza di un algoritmo polinomiale, al punto da basare su questa mancanza una parte fondamentale della tecnologia (essenzialmente, le tecniche crittografiche).

Clausole di Horn

Definizione 5.9

Una clausola (disgiunzione di letterali) è detta di *Horn* se essa contiene al massimo un letterale che non sia una negazione.

Il motivo per considerare queste clausole è che $\neg x_1 \vee \cdots \vee \neg x_n \vee y$ equivale a $x_1 \wedge \cdots \wedge x_n \supset y$, una forma particolarmente conveniente.

Clausole di Horn

Supponiamo che $\phi = \bigwedge_i C_i$ con C_i clausole di Horn. La domanda è: esiste un algoritmo efficiente per stabilire se ϕ è soddisfacibile?

Consideriamo la seguente procedura:

$\forall x.\sigma(x) := \bot$;
while $\exists i.\, c_i \in F \wedge c_i\sigma = \bot \wedge \sigma \neq \top$ do
    let $c_i = (v_1 \wedge \cdots \wedge v_n \supset y)$;
    $\forall x.\sigma(x) :=$ if $x \neq y$ then $\sigma(x)$ else $\top$;
return $\sigma(F)$

Clausole di Horn

$\forall x.\sigma(x) := \bot$;
while $\exists i.\, c_i \in F \wedge c_i\sigma = \bot \wedge \sigma \neq \top$ do
    let $c_i = (v_1 \wedge \cdots \wedge v_n \supset y)$;
    $\forall x.\sigma(x) :=$ if $x \neq y$ then $\sigma(x)$ else $\top$;
return $\sigma(F)$

Questa procedura termina sempre: infatti, $\{x : \sigma(x) = \bot\}$ cresce ad ogni iterazione, e, al peggio, può arrivare al punto in cui tutte le variabili vengono valutate a vero. In tal caso, l'ultima condizione del ciclo impone la terminazione.

Clausole di Horn

$\forall x.\sigma(x) := \bot;$
while $\exists i.\, c_i \in F \wedge c_i\sigma = \bot \wedge \sigma \neq \top$ do
    let $c_i = (v_1 \wedge \cdots \wedge v_n \supset y);$
    $\forall x.\sigma(x) :=$ if $x \neq y$ then $\sigma(x)$ else $\top;$
return $\sigma(F)$

Il risultato dice se l'assegnamento σ soddisfa F, decidendo in modo corretto il problema.

Clausole di Horn

$\forall x.\sigma(x) := \bot$;
while $\exists i.\, c_i \in F \wedge c_i\sigma = \bot \wedge \sigma \neq \top$ do
    let $c_i = (v_1 \wedge \cdots \wedge v_n \supset y)$;
    $\forall x.\sigma(x) :=$ if $x \neq y$ then $\sigma(x)$ else $\top$;
return $\sigma(F)$

Sia τ un assegnamento che soddisfi F. Allora, per ogni variabile x, se $\sigma(x) = \top$ anche $\tau(x) = \top$.
Infatti, questo è vero all'inizio dell'algoritmo, banalmente.
Supponiamo che ad una certa iterazione cessi di esser vero. In questo caso, essendo vero al passo precedente, la clausola c_i che forza y ad esser valutato a vero in σ non può essere soddisfatta da τ, contraddicendo l'assunto.

Clausole di Horn

$\forall x.\sigma(x) := \bot;$
while $\exists i.\, c_i \in F \wedge c_i\sigma = \bot \wedge \sigma \neq \top$ do
    let $c_i = (v_1 \wedge \cdots \wedge v_n \supset y)$;
    $\forall x.\sigma(x) :=$ if $x \neq y$ then $\sigma(x)$ else $\top$;
return $\sigma(F)$

Questo algoritmo esegue in tempo polinomiale rispetto alla dimensione del problema: se F contiene m clausole e n variabili, ad ogni iterazione la clausola c_i viene resa vera, pertanto vi sono, al peggio m iterazioni. Ogni iterazione richiede valutare m clausole per trovarne una non soddisfatta, se esiste. Ogni valutazione richiede valutare n variabili.
Pertanto, la procedura richiede $O(nm^2)$ passi.

Riferimenti

Questa lezione è tratta dal capitolo 4 di *Papadimitriou*, Computational Complexity, Addison-Wesley (1994).

Logica

Lezione 6

Dr Marco Benini

marco.benini@uninsubria.it

Dipartimento di Scienza e Alta Tecnologia
Università degli Studi dell'Insubria

a.a. 2020/21

Syllabus

Logica proposizionale:

- Ordini
- Reticoli
- Algebre di Boole

Ordini

Una semantica più interessante per la logica proposizionale deriva dall'algebra degli ordini.

Definizione 6.1 (Ordine)

Un *ordine* $\mathcal{O} = \langle S; \leq \rangle$ è un insieme S con una relazione binaria $\leq$ che sia

- *riflessiva*, per ogni $x \in S$, $x \leq x$;
- *anti-simmetrica*, per ogni $x, y \in S$, se $x \leq y$ e $y \leq x$, allora $x = y$;
- *transitiva*, per ogni $x, y, z \in S$, se $x \leq y$ e $y \leq z$, allora $x \leq z$.

Ordini

Notando che, se $\mathscr{O} = \langle S; \leq \rangle$ è un ordine, anche $\mathscr{O}^{\mathrm{op}} = \langle S; \geq \rangle$ lo è, otteniamo un *principio di dualità*: quando una proprietà vale per tutti gli ordini, la sua istanza nell'ordine opposto genera una proprietà *duale*, che vale per tutti gli ordini.

Ordini

Definizione 6.2 (Supremo)

Fissato un ordine $\mathscr{O} = \langle S; \leq \rangle$ e un insieme $U \subseteq S$, chiamiamo l'elemento $m \in S$, se esiste, l'*estremo superiore*, o anche il *supremo*, o il *join* di U ogniqualvolta

- per ogni $x \in U$, $x \leq m$;
- per ogni $w \in S$ tale che, se per ogni $x \in U$, $x \leq w$, allora vale che $m \leq w$.

Definizione 6.3 (Infimo)

Fissato un ordine $\mathscr{O} = \langle S; \leq \rangle$ e un insieme $U \subseteq S$, chiamiamo l'elemento $m \in S$, se esiste, l'*estremo inferiore*, o l'*infimo*, o il *meet* di U ogniqualvolta

- per ogni $x \in U$, $m \leq x$;
- per ogni $w \in S$ tale che, se per ogni $x \in U$, $w \leq x$, allora vale che $w \leq m$.

Si osservi come queste nozioni siano duali.

Reticoli

Definizione 6.4 (Reticolo)

Un ordine $\mathscr{O} = \langle S; \leq \rangle$ è detto *reticolo* quando, per ogni coppia $x, y \in S$, esiste il join di $\{x, y\}$, denotato da $x \vee y$, ed esiste il meet di $\{x, y\}$, denotato da $x \wedge y$.

Inoltre, un reticolo è detto essere *limitato* quando, per ogni insieme finito $U \subseteq S$, esiste $\bigvee U$, il join di U, e $\bigwedge U$, il meet di U. Convenzionalmente, $\bigvee \emptyset$ è denotato da $\bot$, *bottom*, e $\bigwedge \emptyset$ è denotato da $\top$, *top*.

Si osservi come i reticoli preservino la dualità.

Enunciato 6.5

In un reticolo limitato $\langle S; \leq \rangle$, ogni elemento è maggiore di $\bot$ e minore di $\top$.

Dimostrazione.

Per dualità, è sufficiente provare solo una metà dell'enunciato.

Poiché $\top = \bigwedge \emptyset$, per definizione di meet, per ogni $x \in \emptyset$, $\top \leq x$, e per ogni $y \in S$ tale che per ogni $x \in \emptyset$, $y \leq x$, vale che $y \leq \top$. Ma non esistono elementi in $\emptyset$, per cui $y \leq \top$ per ogni $y \in S$. □

Enunciato 6.6

In un reticolo limitato $\langle S; \leq \rangle$, $\bigvee S = \top$ *e* $\bigwedge S = \bot$.

Dimostrazione.

Per definizione di join, per ogni $x \in S$, $x \leq \bigvee S$, e, per l'Enunciato 6.5, $\top$ è tale che, per ogni $x \in S$, $x \leq \top$. Quindi $\top \leq \bigvee S$ e $\bigvee S \leq \top$. Per anti-simmetria, $\bigvee S = \top$. L'altra parte segue per dualità. □

Reticoli

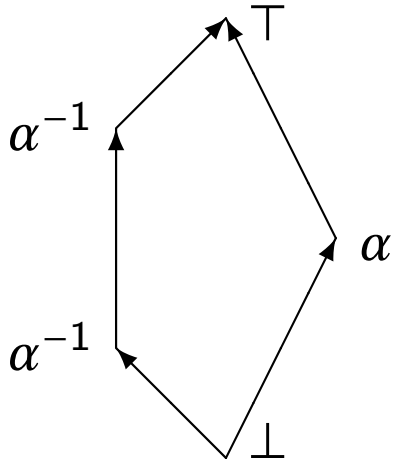
Definizione 6.7 (Reticolo complementato)

Un reticolo limitato $\mathscr{O} = \langle S; \leq \rangle$ è detto essere *complementato* quando, per ogni elemento $x \in S$, esiste un elemento $y \in S$ tale che

- $x \wedge y = \bot$, e
- $x \vee y = \top$.

Reticoli

L'elemento y non è necessariamente unico. Ad esempio:



Reticoli

Definizione 6.8 (Reticolo distributivo)

Un reticolo $\mathscr{O} = \langle S; \leq \rangle$ è detto essere *distributivo* quando, per ogni $x, y, z \in S$, $x \wedge (y \vee z) = (x \wedge y) \vee (x \wedge z)$.

Enunciato 6.9

In ogni reticolo, $x \wedge y = y \wedge x$, *e* $x \vee y = y \vee x$.

Dimostrazione.

Immediata dalla definizione di meet e join. □

Enunciato 6.10

Per ogni x in un reticolo limitato, $x = x \wedge \top$ e $x = x \vee \bot$

Dimostrazione.

Immediata dalla definizione di meet e join, e dall'Enunciato 6.5. □

Enunciato 6.11 (Assorbimento)

Per ogni x e y in un reticolo, $x \vee (x \wedge y) = x$

Dimostrazione.

Per definizione di join, $x \leq x \vee (x \wedge y)$, quindi è sufficiente mostrare che $x \vee (x \wedge y) \leq x$.

Ma $x \leq x$ per riflessività, e $x \wedge y \leq x$ per definizione di meet, quindi $x \vee (x \wedge y) \leq x$ per definizione di join. □

Enunciato 6.12

In qualsiasi reticolo distributivo, per ogni x, y e z,
$x \vee (y \wedge z) = (x \vee y) \wedge (x \vee z)$.

Dimostrazione.

$$
\begin{aligned}
& (x \vee y) \wedge (x \vee z) && \\
= & ((x \vee y) \wedge x) \vee ((x \vee y) \wedge z) && \text{distributività} \\
= & (x \wedge x) \vee (x \wedge y) \vee (x \wedge z) \vee (y \wedge z) && \text{distributività due volte} \\
= & \, x \vee (x \wedge y) \vee (x \wedge z) \vee (y \wedge z) && \text{idempotenza} \\
= & \, x \vee (x \wedge z) \vee (y \wedge z) && \text{assorbimento} \\
= & \, x \vee (y \wedge z) && \text{assorbimento} \quad \square
\end{aligned}
$$

Enunciato 6.13

In ogni reticolo distributivo e complementato, ogni elemento x ha un unico complemento, denotato da $\neg x$.

Dimostrazione.

Supponiamo che l'elemento x abbia due complementi y e z. Allora, per definizione di complemento: $x \wedge y = \bot = x \wedge z$ e $x \vee y = \top = x \vee z$. Perciò

$$
\begin{aligned}
& y \\
&= y \wedge \top \\
&= y \wedge (x \vee z) \\
&= (y \wedge x) \vee (y \wedge z) \\
&= (z \wedge x) \vee (z \wedge y) \\
&= z \wedge (x \vee y) \\
&= z \wedge \top \\
&= z \ .
\end{aligned}
$$

□

Algebre Booleane

Definizione 6.14 (Algebra Booleana)

Un'*algebra Booleana* è un reticolo limitato, distributivo e complementato.

Algebre Booleane

Esempio 6.15

L'insieme $\{0,1\}$, con l'ordinamento $0 \leq 1$, è un'algebra Booleana, con $\top = 1$ e $\bot = 0$. Questa è la struttura che supporta la semantica a tavole di verità.

Esempio 6.16

Fissato un insieme U, l'insieme potenza $\wp(U) = \{S : S \subseteq U\}$ ordinato dall'inclusione, è un'algebra Booleana. Il complemento di S è la differenza $U \setminus S$, mentre $\wedge$ è l'intersezione e $\vee$ è l'unione.

Riferimenti

Le algebre Booleane, nella forma di potenze di un insieme, sono state introdotte per la prima volta in *George Boole*, An Investigation of the Laws of Thought, Prometheus Books, (2003), ristampa dall'edizione originale (1854).

Due ottimi testi di riferimento per gli ordini, i reticoli, e le algebre Booleane sono *B.A. Davey* e *H.A. Priestley*, Introduction to Lattices and Order, Cambridge University Press, (2002), e *George Grätzer*, General Lattice Theory, seconda edizione, Birkhäuser, (1996).

Logica

Lezione 7

Dr Marco Benini

marco.benini@uninsubria.it

Dipartimento di Scienza e Alta Tecnologia
Università degli Studi dell'Insubria

a.a. 2020/21

Syllabus

Logica proposizionale:

- Teorema di correttezza
- Correttezza di programmi semplici

Semantica

Le algebre Booleane sono state introdotte con il preciso scopo di interpretare la logica proposizionale.

Definizione 7.1 (Semantica)

Fissata un'algebra Booleana $\mathscr{O} = \langle O; \leq \rangle$, e una funzione $\nu \colon V \to O$ che mappa ogni variabile in un elemento dell'algebra, l'interpretazione $[\![A]\!]$ di una formula A è definita per induzione come:

- se A è una variabile, $[\![A]\!] = \nu(A)$;
- se $A \equiv \top$, $[\![A]\!] = \top$, il massimo elemento in $\mathscr{O}$;
- se $A \equiv \bot$, $[\![A]\!] = \bot$, il minimo elemento in $\mathscr{O}$;
- se $A \equiv B \wedge C$, $[\![A]\!] = [\![B]\!] \wedge [\![C]\!]$, il meet dell'interpretazione dei congiunti;
- se $A \equiv B \vee C$, $[\![A]\!] = [\![B]\!] \vee [\![C]\!]$, il join dell'interpretazione dei disgiunti;
- se $A \equiv B \supset C$, $[\![A]\!] = \neg[\![B]\!] \vee [\![C]\!]$, ovvero $[\![A]\!] = [\![\neg B \vee C]\!]$, interpretando l'implicazione come un *complemento relativo*;
- se $A \equiv \neg B$, $[\![A]\!] = \neg[\![B]\!]$, il complemento dell'interpretazione di B.

Validità

Definizione 7.2 (Validità)

Una formula A è *valida* o *vera* in un'algebra Booleana $\mathscr{O} = \langle O; \leq \rangle$ con una interpretazione $\nu \colon V \to O$ delle variabili, quando $[\![A]\!] = \top$.
Un insieme di formule è *valido* o *vero* quando ogni formula nell'insieme è valida. La coppia $(\mathscr{O}, \nu)$ è detta un *modello* per la teoria T quando rende vere tutte le formula in T.

Correttezza

Teorema 7.3 (Correttezza)

In ogni modello $\mathcal{O} = (\langle O; \leq \rangle, \nu : V \to O)$ *per la teoria* T *e le assunzioni nell'insieme finito* Δ*, se* $\pi : \Delta \vdash_T A$*, allora* A *è valida.*

Correttezza

Dimostrazione. (i)

La prova è per induzione sulla struttura di π: vogliamo dimostrare che l'interpretazione della conclusione A è maggiore o uguale di $\bigwedge_{G\in\Gamma}[\![G]\!]$, con Γ l'insieme finito di assunzioni che occorrono nella prova di A:

- se π è una dimostrazione per assunzione, allora $A\in\Gamma$ e, per definizione di $\wedge$, $\bigwedge_{G\in\Gamma}[\![G]\!]\leq[\![A]\!]$.
- se π è una dimostrazione per assioma, allora $A\in T$, e, per ipotesi, $[\![A]\!]=\top$, quindi $\bigwedge_{G\in\Gamma}[\![G]\!]\leq[\![A]\!]$ per definizione di $\top$.
- se π è una istanza della Legge del Terzo Escluso, allora $A\equiv B\vee\neg B$, e $[\![A]\!]=[\![B\vee\neg B]\!]=[\![B]\!]\vee\neg[\![B]\!]=\top$ per definizione di complemento in un'algebra Booleana. Perciò $\bigwedge_{G\in\Gamma}[\![G]\!]\leq[\![A]\!]=\top$ per definizione di $\top$.
- se π è una istanza di introduzione di $\top$, allora $A\equiv\top$, quindi $[\![A]\!]=[\![\top]\!]=\top$. Perciò $\bigwedge_{G\in\Gamma}[\![G]\!]\leq[\![A]\!]=\top$ per definizione di $\top$. ↪

Correttezza

↪ Dimostrazione. (ii)

- se π è una istanza di eliminazione di $\bot$, allora, per ipotesi induttiva, $\bot \leq \bigwedge_{G\in\Gamma}[\![G]\!] \leq [\![\bot]\!] = \bot$. Quindi, per anti-simmetria, $\bigwedge_{G\in\Gamma}[\![G]\!] = \bot$. Perciò, per definizione di $\bot$, $\bot = \bigwedge_{G\in\Gamma}[\![G]\!] \leq [\![A[\![$.
- se π è un'istanza di introduzione di $\wedge$, allora $A \equiv B \wedge C$, e per ipotesi induttiva due volte, $\bigwedge_{G\in\Gamma}[\![G]\!] \leq [\![B]\!]$ e $\bigwedge_{G\in\Gamma}[\![G]\!] \leq [\![C]\!]$. Quindi, per definizione di $\wedge$, $\bigwedge_{G\in\Gamma}[\![G]\!] \leq [\![B]\!] \wedge [\![C]\!] = [\![B \wedge C]\!] = [\![A]\!]$.
- se π è un'istanza di $\wedge_1$-eliminazione, allora, per ipotesi induttiva, per qualche formula B, $\bigwedge_{G\in\Gamma}[\![G]\!] \leq [\![A \wedge B]\!] = [\![A]\!] \wedge [\![B]\!]$. Quindi, per definizione di $\wedge$, $\bigwedge_{G\in\Gamma}[\![G]\!] \leq [\![A]\!]$.
- se π è un'istanza di $\wedge_2$-eliminazione, allora, per ipotesi induttiva, per qualche formula B, $\bigwedge_{G\in\Gamma}[\![G]\!] \leq [\![B \wedge A]\!] = [\![B]\!] \wedge [\![A]\!]$. Quindi, per definizione di $\wedge$, $\bigwedge_{G\in\Gamma}[\![G]\!] \leq [\![A]\!]$. ↪

Correttezza

↪ Dimostrazione. (iii)

- se π è un'istanza di $\vee_1$-introduzione, allora $A \equiv B \vee C$ e, per ipotesi induttiva, $\bigwedge_{G \in \Gamma} [\![G]\!] \leq [\![B]\!]$. Quindi, per definizione di $\vee$, $\bigwedge_{G \in \Gamma} [\![G]\!] \leq [\![B]\!] \leq [\![B]\!] \vee [\![C]\!] = [\![B \vee C]\!] = [\![A]\!]$.
- se π è un'istanza di $\vee_2$-introduzione, allora $A \equiv B \vee C$ e, per ipotesi induttiva, $\bigwedge_{G \in \Gamma} [\![G]\!] \leq [\![C]\!]$. Quindi, per definizione di $\vee$, $\bigwedge_{G \in \Gamma} [\![G]\!] \leq [\![C]\!] \leq [\![B]\!] \vee [\![C]\!] = [\![B \vee C]\!] = [\![A]\!]$.
- se π è un'istanza di eliminazione di $\vee$, allora, per ipotesi induttiva, per qualche formula B e C, $\bigwedge_{G \in \Gamma} [\![G]\!] \leq [\![B \vee C]\!] = [\![B]\!] \vee [\![C]\!]$, $[\![B]\!] \wedge \bigwedge_{G \in \Gamma} [\![G]\!] \leq [\![A]\!]$, e $[\![C]\!] \wedge \bigwedge_{G \in \Gamma} [\![G]\!] \leq [\![A]\!]$. Ne segue che, per definizione di $\vee$ e distribuendo, $([\![B]\!] \wedge \bigwedge_{G \in \Gamma} [\![G]\!]) \vee ([\![C]\!] \wedge \bigwedge_{G \in \Gamma} [\![G]\!]) = ([\![B]\!] \vee [\![C]\!]) \wedge \bigwedge_{G \in \Gamma} [\![G]\!] \leq [\![A]\!]$. Ma, poiché $\bigwedge_{G \in \Gamma} [\![G]\!] \leq [\![B]\!] \vee [\![C]\!]$, $([\![B]\!] \vee [\![C]\!]) \wedge \bigwedge_{G \in \Gamma} [\![G]\!] = \bigwedge_{G \in \Gamma} [\![G]\!]$ per definizione di $\wedge$, quindi $\bigwedge_{G \in \Gamma} [\![G]\!] \leq [\![A]\!]$. ↪

Correttezza

↪ Dimostrazione. (iv)

- se π è un'istanza di introduzione di $\supset$, allora $A \equiv B \supset C$ per qualche formula B e C. Per ipotesi induttiva, $[\![B]\!] \wedge \bigwedge_{G \in \Gamma} [\![G]\!] \leq [\![C]\!]$. Quindi, per definizione di $\vee$, $[\![B]\!] \wedge \bigwedge_{G \in \Gamma} [\![G]\!] \leq \neg [\![B]\!] \vee [\![C]\!]$. Evidentemente, $\neg [\![B]\!] \leq \neg [\![B]\!] \vee [\![C]\!]$. Quindi, per definizione di $\vee$, $[\![A]\!] = [\![B \supset C]\!] = \neg [\![B]\!] \vee [\![C]\!] \geq \neg [\![B]\!] \vee ([\![B]\!] \wedge \bigwedge_{G \in \Gamma} [\![G]\!])$. Distribuendo e per definizione di complemento, $[\![A]\!] \geq (\neg [\![B]\!] \vee [\![B]\!]) \wedge (\neg [\![B]\!] \vee \bigwedge_{G \in \Gamma} [\![G]\!]) = \top \wedge (\neg [\![B]\!] \vee \bigwedge_{G \in \Gamma} [\![G]\!]) = \neg [\![B]\!] \vee \bigwedge_{G \in \Gamma} [\![G]\!]$. Per definizione di $\vee$, $\bigwedge_{G \in \Gamma} [\![G]\!] \leq [\![A]\!]$.
- se π è un'istanza di eliminazione di $\supset$, allora, per qualche formula B, per ipotesi induttiva due volte, $\bigwedge_{G \in \Gamma} [\![G]\!] \leq [\![B \supset A]\!]$ e $\bigwedge_{G \in \Gamma} [\![G]\!] \leq [\![B]\!]$. Per definizione di $\wedge$, $\bigwedge_{G \in \Gamma} [\![G]\!] \leq [\![B \supset A]\!] \wedge [\![B]\!]$. Ma $[\![B \supset A]\!] = \neg [\![B]\!] \vee [\![A]\!]$. Quindi, $\bigwedge_{G \in \Gamma} [\![G]\!] \leq (\neg [\![B]\!] \vee [\![A]\!]) \wedge [\![B]\!]$. Distribuendo e per definizione di $\neg$, $\bigwedge_{G \in \Gamma} [\![G]\!] \leq (\neg [\![B]\!] \wedge [\![B]\!]) \vee ([\![A]\!] \wedge [\![B]\!]) = \bot \vee ([\![A]\!] \wedge [\![B]\!]) = [\![A]\!] \wedge [\![B]\!] \leq [\![A]\!]$.

↪

Correttezza

↪ Dimostrazione. (v)

- se π è un'istanza di introduzione di $\neg$, allora $A \equiv \neg B$ per qualche formula B. Quindi, per ipotesi induttiva, $[\![B]\!] \wedge \bigwedge_{G \in \Gamma} [\![G]\!] \leq [\![\bot]\!] = \bot$. Perciò, per definizione di $\bot$ e anti-simmetria, $[\![B]\!] \wedge \bigwedge_{G \in \Gamma} [\![G]\!] = \bot$. Quindi, $[\![A]\!] = [\![\neg B]\!] = \neg [\![B]\!] = \neg [\![B]\!] \vee \bot = \neg [\![B]\!] \vee ([\![B]\!] \wedge \bigwedge_{G \in \Gamma} [\![G]\!])$, e distribuendo, $[\![A]\!] = (\neg [\![B]\!] \vee [\![B]\!]) \wedge (\neg [\![B]\!] \vee \bigwedge_{G \in \Gamma} [\![G]\!]) = \top \wedge ([\![\neg B]\!] \vee \bigwedge_{G \in \Gamma} [\![G]\!]) = [\![A]\!] \vee \bigwedge_{G \in \Gamma} [\![G]\!]$. Quindi, per definizione di $\vee$, $\bigwedge_{G \in \Gamma} [\![G]\!] \leq [\![A]\!]$.
- se π è un'istanza di eliminazione di $\neg$, allora $A \equiv \bot$ e, per ipotesi induttiva due volte, $\bigwedge_{G \in \Gamma} [\![G]\!] \leq [\![\neg B]\!]$ e $\bigwedge_{G \in \Gamma} [\![G]\!] \leq [\![B]\!]$. Ma $[\![\neg B]\!] = \neg [\![B]\!]$. Quindi, per definizione di $\wedge$, $\bigwedge_{G \in \Gamma} [\![G]\!] \leq \neg [\![B]\!] \wedge [\![B]\!]$. Per definizione di complemento, $\bigwedge_{G \in \Gamma} [\![G]\!] \leq \neg [\![B]\!] \wedge [\![B]\!] = \bot = [\![A]\!]$.

Quindi, per ogni formula A che sia la conclusione di una dimostrazione da Δ nella teoria T, $\bigwedge_{G \in \Delta} [\![G]\!] \leq [\![A]\!]$. Ma, per ipotesi, per ogni $G \in \Delta$, $[\![G]\!] = \top$, quindi $\bigwedge_{G \in \Delta} [\![G]\!] = \top$, perciò, per definizione di $\top$, $\top \leq [\![A]\!] \leq \top$, ovvero, per anti-simmetria, $[\![A]\!] = \top$. □

Correttezza di programmi

L'idea di fondo della dimostrazione del Teorema di Correttezza può essere trasferita sui programmi.

Se immaginiamo un programma come una dimostrazione della proprietà che vuole calcolare, partendo dalle assunzioni sul suo input, possiamo verificare che tale dimostrazione è corretta, mostrando che, in effetti, il programma calcola ciò che intende.

L'idea è che ogni istruzione deve essere vista come una istanza di una regola di inferenza.

Correttezza di programmi

Consideriamo la seguente procedura:

binarySearch(A,k) =
$n := \text{length}(A)$;
if $n \leq 0$ then return *Key not found*;
$i := n/2$;
if $A[i].\text{key} = k$ then return i;
else
if $A[i].\text{key} > k$ then binarySearch$(A[1 \ldots (i-1)], k)$;
else binarySearch$(A[(i+1) \ldots n], k)$.

Essa, dato un array ordinato e una chiave, trova quale elemento dell'array sia uguale alla chiave, se essa è presente.

Correttezza di programmi

Quanto vogliamo dimostrare è che, se A è ordinato,

$$\forall 1 \leq i < j \leq n. A[i].\text{key} \leq A[j].\text{key} \quad ,$$

con n pari alla lunghezza dell'array, allora vale che il risultato è

- *Key not found* e $\forall 1 \leq i \leq n. A[i].\text{key} \neq k$,
- h con $A[h].\text{key} = k$.

Procediamo per induzione sulla computazione.

Correttezza di programmi

binarySearch(A,k) =
$n := \text{length}(A)$;
if $n \leq 0$ then return *Key not found*;
$i := n/2$;
if $A[i].\text{key} = k$ then return i;
else
if $A[i].\text{key} > k$ then binarySearch($A[1 \ldots (i-1)], k$);
else binarySearch($A[(i+1) \ldots n], k$).

Chiaramente, se A è vuoto, allora il risultato è *Key not found* e nessun elemento di A ha chiave pari a k.

Correttezza di programmi

binarySearch(A,k) =

$n :=$ length(A);
if $n \leq 0$ then return *Key not found*;
$i := n/2$;
if $A[i]$.key $= k$ then return i;
else
  if $A[i]$.key $> k$ then binarySearch$(A[1 \ldots (i-1)], k)$;
  else binarySearch$(A[(i+1) \ldots n], k)$.

Altrimenti, sappiamo che

- A non è vuoto, quindi contiene un elemento in posizione media,
- sia i tale elemento

Correttezza di programmi

binarySearch(A,k) =
$n :=$ length(A);
if $n \leq 0$ then return *Key not found*;
$i := n/2$;
if $A[i]$.key $= k$ then return i;
else
if $A[i]$.key $> k$ then binarySearch$(A[1 \ldots (i-1)], k)$;
else binarySearch$(A[(i+1) \ldots n], k)$.

Se $A[i]$.key $= k$ allora il risultato è i e vale banalmente l'asserto di correttezza.

Correttezza di programmi

binarySearch(A,k) =

$n :=$ length(A);
if $n \leq 0$ then return *Key not found*;
$i := n/2$;
if $A[i]$.key $= k$ then return i;
else
  if $A[i]$.key $> k$ then binarySearch$(A[1 \ldots (i-1)], k)$;
  else binarySearch$(A[(i+1) \ldots n], k)$.

Altrimenti, se $A[i]$.key $> k$, essendo l'array ordinato, se presente, k occorre nel sotto-array da 1 a $i-1$, e per ipotesi induttiva, la procedura ottiene il risultato corretto.

Correttezza di programmi

binarySearch(A,k) =

$n :=$ length(A);
if $n \leq 0$ then return *Key not found*;
$i := n/2$;
if $A[i]$.key $= k$ then return i;
else
  if $A[i]$.key $> k$ then binarySearch$(A[1 \ldots (i-1)], k)$;
  else binarySearch$(A[(i+1) \ldots n], k)$.

Oppure $A[i]$.key $< k$, e, per lo stesso motivo, se presente la chiave occorre nel sotto-array da $i+1$ a n e, ancora, per ipotesi induttiva, la procedura ottiene il risultato corretto.

Correttezza di programmi

La parte saliente di questa prova di correttezza è il ragionamento per induzione, in cui ogni passaggio usa l'istruzione da eseguire come se fosse una regola di inferenza che trasforma lo stato della computazione.

Questa idea verrà esplorata nel prosieguo del corso.

Riferimenti

L'idea della prova del Teorema di Correttezza è folklore: infatti, la prova presentata è stata adattata da un risultato più generale che usa la logica interna di un topos Booleano. Questo è un argomento avanzato che non verrà trattato in questo corso. Lo studente interessato può dare un'occhiata a *P. Johnstone*, Sketches of an Elephant: A Topos Theory Compendium, due volumi, Oxford University Press (2002).

Riferimenti

Il tema della correttezza dei programmi è stato esplorato in innumerevoli approcci, e con tantissime tecniche differenti.

L'esempio riportato è tratto dalla tesi di dottorato del docente: *M. Benini*, Verification and Analysis of Programs in a Constructive Environment, Dottorato di Ricerca in Informatica, Università degli Studi di Milano (1999)

Logica

Lezione 8

Dr Marco Benini

marco.benini@uninsubria.it

Dipartimento di Scienza e Alta Tecnologia
Università degli Studi dell'Insubria

a.a. 2020/21

Syllabus

Logica proposizionale:

- Completezza

Completezza

Quanto vogliamo dimostrare in questa lezione è che, fissata una teoria T, per ogni insieme finito Γ di formule e per ogni formula A, se $\bigwedge_{G \in \Gamma} [\![G]\!] \leq [\![A]\!]$ vale in ogni algebra Booleana che renda vera la teoria T, allora esiste una deduzione in calcolo naturale tale che $\pi : \Gamma \vdash_T A$.

Come corollario, notando che quando $\Gamma = \emptyset$, $\bigwedge_{G \in \Gamma} [\![G]\!] = \top$, segue che ogni formula A che sia vera in ogni algebra Booleana che renda vera la teoria, è anche dimostrabile.

Preliminari

La dimostrazione è complessa e sottile.

In primo luogo, notiamo che, se $\pi : \Gamma \vdash_T A$, allora esiste un sottoinsieme finito $\Delta \subseteq \Gamma$ tale che $\pi : \Delta \vdash_T A$. Infatti, poiché ogni prova è un oggetto finito e ogni regola d'inferenza ha un numero finito di antecedenti, soltanto un numero finito di assunzioni può comparire in una dimostrazione formale.

In questo senso, il limite di avere un enunciato per il Teorema di Completezza in cui si richieda che Γ sia finito non è limitante.

Strategia

La parte difficile della dimostrazione del Teorema di Completezza consiste nel considerare la totalità delle algebre Booleane.

La strategia della dimostrazione richiede costruire un'algebra Booleana *canonica* $\mathbb{B}$ che renda veri gli assiomi di T, ed in cui il concetto di vero e dimostrabile coincidano.
Questa strategia è generale, e risultati di completezza in altri sistemi formali seguono lo stesso metodo. Tuttavia vi sono eccezioni, e sistemi in cui tale strategia non può riuscire.

Modello canonico

Definizione 8.1 (Algebra Booleana canonica)

Sia T una teoria. L'*algebra Booleana canonica* $\mathbb{B}(T)$ su T è l'insieme $\{A\colon A \text{ è una formula nel linguaggio di } T\}/\sim$, dove

- $A \sim B$ se e solo se $A \vdash_T B$ e $B \vdash_T A$,
- $[A]_\sim \leq_{\mathbb{B}(\mathbb{T})} [B]_\sim$ esattamente quando $A \vdash_T B$.

Per semplicità, quando è chiaro dal contesto, omettiamo i pedici. Anche, si osservi come $[A]_\sim \leq_{\mathbb{B}(\mathbb{T})} [B]_\sim$ e $[B]_\sim \leq_{\mathbb{B}(\mathbb{T})} [A]_\sim$ implicano $[A]_\sim = [B]_\sim$.

Si noti come, ponendo A essere vera esattamente quando $[\top]_\sim \leq [A]_\sim$, otteniamo che $\emptyset \vdash A$ perché $\top = \bigwedge \emptyset$.

Ma, prima, dobbiamo dimostrare che $\mathbb{B}(T)$ è un'algebra Booleana.

Un risultato ausiliario

Lemma 8.2

Se $\pi : \Gamma \cup \{A\} \vdash_T B$ *e* $\theta : \Gamma \vdash_T A$*, allora esiste una deduzione* $\nu : \Gamma \vdash_T B$*.*

Dimostrazione. (i)

Per induzione sulla dimostrazione π.

- se π è un'istanza della regola di assunzione, allora $B \in \Gamma$, e quindi ν coincide con π che non dipende da A, oppure $B \equiv A$, e quindi $\nu = \theta$.
- se π è un'istanza della regola di assioma, $B \in T$, quindi $\nu = \pi$, che non dipende da A.
- se π è un'istanza di introduzione di $\top$, $B \equiv \top$, quindi $\nu = \pi$, che non dipende da A.
- se π è un'istanza di eliminazione di $\bot$, per ipotesi induttiva esiste $\xi : \Gamma \vdash_T \bot$, quindi applicando la regola di eliminazione di $\bot$ a ξ otteniamo la prova ν. ↪

Un risultato ausiliario

↪ Dimostrazione. (ii)

- se π è un'istanza della Legge del Terzo Escluso, $B \equiv C \vee \neg C$, quindi $\nu = \pi$, che non dipende da A.
- se π è un'istanza di introduzione di $\wedge$, $B \equiv C \wedge D$ e, per ipotesi induttiva, ci sono $\xi : \Gamma \vdash_T C$ e $\mu : \Gamma \vdash_T D$, quindi ν si ottiene per introduzione di $\wedge$ da ξ e μ.
- se π è un'istanza di eliminazione di $\wedge$ a sinistra, per ipotesi induttiva esiste $\xi : \Gamma \vdash_T B \wedge C$, quindi ν si ottiene applicando a ξ la regola di eliminazione di $\wedge$ a sinistra.
- se π è un'istanza di eliminazione di $\wedge$ a destra, per ipotesi induttiva esiste $\xi : \Gamma \vdash_T B \wedge C$, quindi ν si ottiene applicando a ξ la regola di eliminazione di $\wedge$ a destra. ↪

Un risultato ausiliario

↪ Dimostrazione. (iii)

- se π è un'istanza di introduzione di $\vee$ a sinistra, allora $B \equiv C \vee D$ e, per ipotesi induttiva esiste $\xi : \Gamma \vdash_T C$, quindi ν si ottiene applicando a ξ la regola di introduzione di $\vee$ a sinistra.
- se π è un'istanza di introduzione di $\vee$ a destra, allora $B \equiv C \vee D$ e, per ipotesi induttiva esiste $\xi : \Gamma \vdash_T D$, quindi ν si ottiene applicando a ξ la regola di introduzione di $\vee$ a destra.
- se π è un'istanza di eliminazione di $\vee$, per ipotesi induttiva ci sono $\xi : \Gamma \vdash_T C \vee D$, $\mu_C : \Gamma \cup \{C\} \vdash_T B$ e $\mu_D : \Gamma \cup \{D\} \vdash_T B$, quindi applicando l'eliminazione di $\vee$ a ξ, μ_C e μ_D si costruisce ν. ↪

Un risultato ausiliario

↪ Dimostrazione. (iv)

- se π è un'istanza di introduzione di $\supset$, allora $B \equiv C \supset D$ e, per ipotesi induttiva esiste $\xi : \Gamma \cup \{C\} \vdash_T D$, quindi ν si ottiene da ξ applicando una introduzione di $\supset$.
- se π è un'istanza di eliminazione di $\supset$, per ipotesi induttiva esistono $\xi : \Gamma \vdash_T C \supset B$ e $\mu : \Gamma \vdash_T C$, quindi ν si costruisce applicando l'eliminazione di $\supset$ a ξ e μ.
- se π è un'istanza di introduzione di $\neg$, $B \equiv \neg C$ e, per ipotesi induttiva esiste $\xi : \Gamma \cup \{C\} \vdash_T \bot$, e ν si ottiene applicando a ξ l'introduzione di $\neg$.
- se π è un'istanza di eliminazione di $\neg$, per ipotesi induttiva esistono $\xi : \Gamma \vdash_T \neg C$ e $\mu : \Gamma \vdash_T C$, quindi ν si costruisce applicando l'eliminazione di $\neg$ a ξ e μ. □

Proprietà del modello canonico

Enunciato 8.3

La relazione $\sim$ è una relazione di equivalenza.

Dimostrazione.

- Per la regola d'inferenza di assunzione, $A \vdash_T A$, quindi $A \sim A$ per ogni formula A, ovvero $\sim$ è riflessiva.
- Se $A \sim B$, allora $A \vdash_T B$ e $B \vdash_T A$, quindi $B \sim A$. Ovvero $\sim$ è simmetrica.
- Se $A \sim B$ e $B \sim C$ allora ci sono $\pi_B \colon A \vdash_T B$ e $\pi_A \colon B \vdash_T A$, e $\theta_C \colon B \vdash_T C$ e $\theta_B \colon C \vdash_T B$. Per il Lemma 8.2, esistono $\pi \colon A \vdash_T C$ e $\theta \colon C \vdash_T A$, ovvero $A \sim C$, che significa che $\sim$ è transitiva. □

Proprietà del modello canonico

Enunciato 8.4

La relazione $\leq_{\mathbb{B}(T)}$ è un ordine.

Dimostrazione.

- La relazione $[A]_\sim \leq [B]_\sim$ non dipende dalla scelta dei rappresentanti nelle classi di equivalenza su $\sim$. Infatti, se $[A]_\sim = [A']_\sim$ e $[B]_\sim = [B']_\sim$, allora $A \sim A'$ e $B \sim B'$. Quindi, per definizione di $\sim$, $A' \vdash_T A$ e $B \vdash_T B'$. Ma, per definizione di $\leq$, $A \vdash_T B$, quindi per il Lemma 8.2 due volte, $A' \vdash_T B'$, ovvero $[A']_\sim \leq [B']_\sim$.
- Per la regola di assunzione, $A \vdash_T A$, quindi $[A]_\sim \leq [A]_\sim$, ovvero $\leq$ è riflessiva.
- Se $[A]_\sim \leq [B]_\sim$ e $[B]_\sim \leq [C]_\sim$ allora $A \vdash_T B$ e $B \vdash_T C$, quindi per il Lemma 8.2, $A \vdash_T C$, ovvero $[A]_\sim \leq [C]_\sim$, che significa che $\leq$ è transitiva.
- Se $[A]_\sim \leq [B]_\sim$ e $[B]_\sim \leq [A]_\sim$ allora $A \vdash_T B$ e $B \vdash_T A$, quindi per definizione di $\sim$, $A \sim B$, ovvero, $[A]_\sim = [B]_\sim$, che significa che $\leq$ è anti-simmetrica. □

Proprietà del modello canonico

Enunciato 8.5

$\mathbb{B}(T)$ *è un reticolo.*

Dimostrazione.

- Si consideri $[A \wedge B]_\sim$: $[A \wedge B]_\sim \leq [A]_\sim$ poiché $A \wedge B \vdash_T A$ per $\wedge_1$-eliminazione; anche, $[A \wedge B]_\sim \leq [B]_\sim$ poiché $A \wedge B \vdash_T B$ per $\wedge_2$-eliminazione. Se $[C]_\sim \leq [A]_\sim$ e $[C]_\sim \leq [B]_\sim$, allora $C \vdash_T A$ e $C \vdash_T B$, quindi $C \vdash_T A \wedge B$ per $\wedge$-introduzione, perciò $[C]_\sim \leq [A \wedge B]_\sim$. Quindi, per definizione di $\wedge$ in un ordine, $[A]_\sim \wedge [B]_\sim = [A \wedge B]_\sim$.
- Si consideri $[A \vee B]_\sim$: $[A]_\sim \leq [A \vee B]_\sim$ poiché $A \vdash_T A \vee B$ per $\vee_1$-introduzione; anche, $[B]_\sim \leq [A \vee B]_\sim$ poiché $B \vdash_T A \vee B$ per $\vee_2$-introduzione. Se $[A]_\sim \leq [C]_\sim$ e $[B]_\sim \leq [C]_\sim$, allora $A \vdash_T C$ e $B \vdash_T C$, quindi $A \vee B \vdash_T C$ per $\vee$-eliminazione, pertanto $[A \vee B]_\sim \leq [C]_\sim$. Quindi, per definizione di $\vee$ in un ordine, $[A]_\sim \vee [B]_\sim = [A \vee B]_\sim$. □

Proprietà del modello canonico

Enunciato 8.6

$\mathbb{B}(T)$ *è un reticolo limitato.*

Dimostrazione.

- Per ogni formula A, $A \vdash_T \top$ per $\top$-introduzione, quindi $[A]_\sim \leq [\top]_\sim$. Quindi, per definizione di $\top$ in un reticolo, $\top = [\top]_\sim$.
- Per ogni formula A, $\bot \vdash_T A$ per $\bot$-eliminazione, quindi $[\bot]_\sim \leq [A]_\sim$. Pertanto, per definizione di $\bot$ in un reticolo, $\bot = [\bot]_\sim$. □

Proprietà del modello canonico

Enunciato 8.7

$\mathbb{B}(T)$ *è un reticolo distributivo.*

Dimostrazione. (i)

Per ogni A, B e C, $[A] \vee ([B] \wedge [C]) = [A] \vee [B \wedge C] = [A \vee (B \wedge C)]$ e $([A] \vee [B]) \wedge ([A] \vee [C]) = [A \vee B] \wedge [A \vee C] = [(A \vee B) \wedge (A \vee C)]$.

Ma $A \vee (B \wedge C) \vdash_T (A \vee B) \wedge (A \vee C)$ poiché

$$\dfrac{A \vee (B \wedge C) \quad \dfrac{\dfrac{[A]^*}{A \vee B} \vee \mathrm{I}_1 \quad \dfrac{[A]^*}{A \vee C} \vee \mathrm{I}_1}{(A \vee B) \wedge (A \vee C)} \wedge \mathrm{I} \quad \dfrac{\dfrac{\dfrac{[B \wedge C]^*}{B} \wedge \mathrm{E}_1}{A \vee B} \vee \mathrm{I}_2 \quad \dfrac{\dfrac{[B \wedge C]^*}{C} \wedge \mathrm{E}_2}{A \vee C} \vee \mathrm{I}_2}{(A \vee B) \wedge (A \vee C)} \wedge \mathrm{I}}{(A \vee B) \wedge (A \vee C)} \vee \mathrm{E}^*$$

↪

Proprietà del modello canonico

↪ Dimostrazione. (ii)

Anche $(A \vee B) \wedge (A \vee C) \vdash_T A \vee (B \wedge C)$ poiché

$$\dfrac{\dfrac{(A \vee B) \wedge (A \vee C)}{A \vee B}\,\wedge E_1 \quad \dfrac{[A]^*}{A \vee (B \wedge C)}\,\vee I_1 \quad \begin{array}{c}[B]^* \\ \vdots \\ A \vee (B \wedge C)\end{array}}{A \vee (B \wedge C)}\,\vee E^*$$

dove il terzo antecedente è

$$\dfrac{\dfrac{(A \vee B) \wedge (A \vee C)}{A \vee C}\,\wedge E_2 \quad \dfrac{[A]^\dagger}{A \vee (B \wedge C)}\,\vee I_1 \quad \dfrac{\dfrac{B \quad [C]^\dagger}{B \wedge C}\,\wedge I}{A \vee (B \wedge C)}\,\vee I_2}{A \vee (B \wedge C)}\,\vee E^\dagger$$

Quindi $(A \vee B) \wedge (A \vee C) \sim A \vee (B \wedge C)$. □

Proprietà del modello canonico

Enunciato 8.8

$\mathbb{B}(T)$ *è un reticolo complementato.*

Dimostrazione.

Si consideri, per qualsiasi formula A, $[\neg A]$: $[A] \wedge [\neg A] = [A \wedge \neg A] = [\bot] = \bot$, poiché $\bot \vdash_T A \wedge \neg A$ per $\bot$-eliminazione, e

$$\dfrac{\dfrac{A \wedge \neg A}{A}\,\wedge E_1 \qquad \dfrac{A \wedge \neg A}{\neg A}\,\wedge E_2}{\bot}\,\neg E$$

Inoltre, $[A] \vee [\neg A] = [A \vee \neg A] = [\top] = \top$, poiché $A \vee \neg A \vdash_T \top$ per $\top$-introduzione, e $\top \vdash_T A \vee \neg A$ per la Legge del Terzo Escluso. □

Corollario 8.9

$\mathbb{B}(T)$ *è un'algebra Booleana.*

Completezza

Teorema 8.10 (Completezza)

Fissata una teoria T, per ogni insieme finito Γ di formule e per ogni formula A, se $\bigwedge_{G\in\Gamma}[\![G]\!] \leq [\![A]\!]$ in ogni modello per T, allora esiste una prova in calcolo naturale $\pi : \Gamma \vdash_T A$.

Dimostrazione.

Se $\bigwedge_{G\in\Gamma}[\![G]\!] \leq [\![A]\!]$ allora $[\![\bigwedge_{G\in\Gamma} G]\!] \leq [\![A]\!]$ essendo Γ finito.

Poiché questo fatto vale in ogni algebra Booleana, vale in particolare in $\mathbb{B}(T)$, l'algebra Booleana canonica su T. E per come è stata definita l'interpretazione in $\mathbb{B}(T)$, $[\bigwedge_{G\in\Gamma} G] \leq [A]$.

Quindi, per definizione di $\leq$ in $\mathbb{B}(T)$, esiste $\pi : \bigwedge_{G\in\Gamma} G \vdash_T A$. Si noti che $\Gamma \vdash_T \bigwedge_{G\in\Gamma} G$ iterando la regola di $\wedge$-introduzione. Quindi, per il Lemma 8.2 segue $\Gamma \vdash_T A$. □

Completeness

Corollario 8.11

Se $[\![A]\!] = \top$ *in ogni modello per* T*, allora esiste una dimostrazione* $\pi \colon \vdash_T A$*.*

Dimostrazione.

Se $[\![A]\!] = \top$ allora $\top = [\![\top]\!] \leq [\![A]\!]$, essendo $\leq$ riflessiva. Per il Teorema di Completezza, il risultato consegue immediatamente. □

Riferimenti

La dimostrazione è stata adattata da quella relativa alla teoria dei topos, illustrata nella Sezione D di *Peter Johnstone*, Sketches of an Elephant: A Topos Theory Compendium, Oxford Logic Guides 43, Oxford University Press, (2003).

Logica

Lezione 9

Dr Marco Benini

marco.benini@uninsubria.it

Dipartimento di Scienza e Alta Tecnologia
Università degli Studi dell'Insubria

a.a. 2020/21

Syllabus

Logica al prim'ordine:

- Sintassi
- Sostituzione
- Calcolo naturale

Logica al primo ordine

La logica proposizionale è un sistema molto utile, ma non possiede abbastanza potenza espressiva per descrivere una teoria matematica significativa, ad esempio, l'aritmetica o la teoria degli insiemi, e quindi è molto limitata nelle applicazioni.

Sebbene la logica proposizionale e le sue teorie siano strutturalmente ben fatte, se vogliamo usare la logica come strumento per descrivere matematica significativa, abbiamo bisogno di parlare degli oggetti.

La principale novità della logica al primo ordine è che il linguaggio è capace di identificare sintatticamente gli oggetti, e di scrivere formule riguardo ad essi. Come già detto, permettiamo la quantificazione sugli oggetti, ma non su insiemi o su altre strutture di oggetti.

Sintassi

Definizione 9.1 (Segnatura)

Una *segnatura* $\Sigma = \langle S; F; R \rangle$ è composta da

- un insieme S di simboli per *sorte*.
- un insieme F di simboli per *funzioni*. Ogni simbolo $f \in F$ è univocamente associato ad un *tipo* $s_1 \times \cdots \times s_n \to s_0$, con $s_i \in S$ per ogni $0 \leq i \leq n$. Quando $n = 0$, diciamo che f è una *costante* di tipo s_0.
- un insieme R di simboli per *relazioni*. Ogni simbolo $r \in R$ è univocamente associato ad un *tipo* $s_1 \times \cdots \times s_n$, con $s_i \in S$ per ogni $1 \leq i \leq n$. Quando $n = 0$, diciamo che r è una *costante proposizionale*.

La notazione $f : s_1 \times \cdots \times s_n \to s_0 \in F$ e $r : s_1 \times \cdots \times s_n \in R$ indica che f è un simbolo di funzione il cui tipo è $s_1 \times \cdots \times s_n \to s_0$, e r è un simbolo di relazione il cui tipo è $s_1 \times \cdots \times s_n$. Inoltre richiediamo che S, F e R non contengano come simboli i connettivi logici e i quantificatori.

Una segnatura descrive un linguaggio al prim'ordine: le sorte stanno per le collezioni di elementi, le funzioni sono usate per denotare gli elementi singoli, e le relazioni sono usate per costruire le formule di base.

Esempio 9.2

La segnatura

$$\mathscr{N} = \langle \{\mathbb{N}\}; \{0 : \mathbb{N}, S : \mathbb{N} \to \mathbb{N}; + : \mathbb{N} \times \mathbb{N} \to \mathbb{N}, \cdot : \mathbb{N} \times \mathbb{N} \to \mathbb{N}\}; \{= : \mathbb{N} \times \mathbb{N}\} \rangle$$

specifica il linguaggio per l'aritmetica. Vi è una sola sorta, che sta per i numeri naturali. C'è una costante, 0 che denota lo zero, una funzione S che denota il "successore", e le funzioni $+$ e $\cdot$ che stanno per la somma e il prodotto.
Vi è una sola relazione, $=$, che denota l'uguaglianza.

Sintassi

Esempio 9.3
La segnatura $\mathscr{G} = \langle \{G\}; \{1\colon G, \cdot\colon G \times G \to G, \_^{-1}\colon G \to G\}; \{=\colon G \times G\}\rangle$ descrive il linguaggio della teoria dei gruppi.

Esempio 9.4
La segnatura $\mathscr{O} = \langle \{O\}; \emptyset; \{\leq\colon O \times O\}\rangle$ descrive il linguaggio della teoria degli ordini.

Esempio 9.5
La segnatura $\mathscr{L} = \langle \{E, L\}; \{\mathsf{nil}\colon L, \mathsf{cons}\colon E \times L \to L\}; \{=_E\colon E \times E, =_L\colon L \times L\}\rangle$ definisce il linguaggio per la teoria delle liste. Un informatico potrebbe dire che definisce il linguaggio per il *tipo di dato astratto* delle liste.

Termini

Il linguaggio al prim'ordine ha due scopi: fornire una sintassi per denotare gli elementi nelle sorte, e denotare le proprietà di questi elementi.

I *termini* soddisfano la prima esigenza.

Definizione 9.6 (Termine)

Sia $\Sigma = \langle S; F; R \rangle$ una segnatura, e sia V un insieme infinito di simboli, dette *variabili*, tale che $V \cap (S \cup F \cup R) = \emptyset$. Inoltre, assumiamo che ogni variabile $x \in V$ sia univocamente associata ad un tipo $s \in S$, denotato da $x : s$.
Richiediamo che vi siano infinite variabili per ogni tipo $s \in S$.
Un *termine*, assieme al suo insieme di *variabili libere*, è definito per induzione come segue:

- se $x : s \in V$, allora x è un termine di tipo s, e $\mathrm{FV}(x) = \{x\}$;
- se $f : s_1 \times \cdots \times s_n \to s_0 \in F$ e $t_1, \ldots, t_n$ sono termini di tipo $s_1, \ldots, s_n$, rispettivamente, allora $f(t_1, \ldots, t_n)$ è un termine di tipo s_0 e $\mathrm{FV}(f(t_1, \ldots, t_n)) = \bigcup_{i=1}^{n} \mathrm{FV}(t_i)$.

Usiamo la notazione $t : s$ per dire che il termine t ha tipo s.

Termini

Esempio 9.7

Usando la segnatura $\mathscr{N}$ dell'aritmetica, 0, $S(0)$, $S(S(0))$, ... sono termini di tipo $\mathbb{N}$. Anche $+(x,0)$ e $\cdot(x,+(S(0),S(S(0))))$ sono termini di tipo $\mathbb{N}$. Si noti come $x+0$ e $x(1+2)$ **non** sono termini.

Formule

Così come i termini sono usati per denotare gli elementi, le formule sono usate per denotare le proprietà degli elementi. La sintassi è molto simile alla logica proposizionale, con due importanti differenze: vi sono le *formule atomiche* al posto delle variabili proposizionali, e vi sono i *quantificatori*.

Definizione 9.8 (Formula)

Fissata una segnatura $\Sigma = \langle S;\ F; R\rangle$ e un insieme di variabili come per i termini, una *formula*, assieme al suo insieme di *variabili libere*, è definita per induzione come

- $\top$ e $\bot$ sono formule, e $\mathrm{FV}(\top) = \mathrm{FV}(\bot) = \emptyset$.
- se $r : s_1 \times \cdots \times s_n \in R$ è un simbolo di relazione, e $t_1 : s_1, \ldots, t_n : s_n$ sono termini, allora $r(t_1, \ldots, t_n)$ è una formula *atomica*, e $\mathrm{FV}\left(r(t_1, \ldots, t_n)\right) = \bigcup_{i=1}^{n} \mathrm{FV}(t_i)$.
- se A e B sono formule, anche $\neg A$, $A \wedge B$, $A \vee B$ e $A \supset B$ lo sono, e $\mathrm{FV}(\neg A) = \mathrm{FV}(A)$, $\mathrm{FV}(A \wedge B) = \mathrm{FV}(A \vee B) = \mathrm{FV}(A \supset B) = \mathrm{FV}(A) \cup \mathrm{FV}(B)$.
- se $x : s$ è una variabile e A è una formula, allora $\forall x : s.\, A$ e $\exists x : s.\, A$ sono formule, e $\mathrm{FV}(\forall x : s.\, A) = \mathrm{FV}(\exists x : s.\, A) = \mathrm{FV}(A) \setminus \{x\}$.

Formule

Vi sono due importanti differenze tra le formule al prim'ordine e le formule proposizionali:

- invece delle variabili proposizionali abbiamo le formule atomiche, che collegano le formule con i termini mediante le relazioni,
- vi sono le formule quantificate, dove la variabile **non** è libera. Diciamo che le variabili quantificate sono *legate*.

La nozione di variabile legata non è nuova: per esempio, l'espressione $\int_a^b f(x)\,\mathrm{d}x$ in Analisi non dipende realmente dalla variabile x. Infatti, la x è un segnaposto per dare un nome all'argomento della funzione f.
Una variabile legata non denota un valore, ma agisce come un segnaposto che permette di scrivere una formula o un termine. Il suo significato è controllato dal quantificatore, e non dal modo in cui andiamo ad interpretare le variabili, come nell'integrale la x non denota un numero reale, ma piuttosto cosa permettiamo di variare nella funzione.

Sostituzione

Le variabili sono soggetto di una operazione fondamentale: la *sostituzione*. Infatti, dalla formula A dove la variabile x compare libera, possiamo ottenere un'altra formula, $A[t/x]$, dove il termine t sostituisce x. Per esempio, nel linguaggio dell'aritmetica, x può essere sostituita da 2 in $x+0=x$ per ottenere $2+0=2$.

La sostituzione è fondamentale nel descrivere le regole di inferenza che governano la quantificazione. E le variabili legate rendono la sostituzione una operazione non immediatamente intuitiva.

Ci sono molti modi equivalenti di definire l'operazione di sostituzione: noi adotteremo un metodo che non è il più immediato, ma che diventerà molto comodo più avanti nel corso.

Sostituzione

Definizione 9.9 (Sostituzione sui termini)

Fissata una segnatura e un termine t in essa, la *sostituzione* della variabile $x : s$ con il termine $r : s$, che produce $t[r/x]$, è definita per induzione sulla struttura del termine t:

- se $t \equiv x$, allora $t[r/x] = r$;
- se t è una variabile $t \not\equiv x$, allora $t[r/x] = t$;
- se $t \equiv f(t_1, \ldots, t_n)$, allora $t[r/x] = f(t_1[r/x], \ldots, t_n[r/x])$.

Si noti come l'operazione di sostituzione è definita solamente quando r e x hanno lo stesso tipo.

Sostituzione

Definizione 9.10 (Sostituzione sulle formule)

Fissata una segnature e una formula A su di essa, la *sostituzione* della variabile $x : s$ con il termine $t : s$, che produce $A[t/x]$, è definita per induzione sulla struttura della formula A:

- se $A \equiv \top$ o $A \equiv \bot$, allora $A[t/x] = A$;
- se $A \equiv r(t_1, \ldots, t_n)$, allora $A[t/x] = r(t_1[t/x], \ldots, t_n[t/x])$;
- se $A \equiv \neg B$, allora $A[t/x] = \neg B[t/x]$;
- se $A \equiv B \wedge C$, $A \equiv B \vee C$, o $A \equiv B \supset C$, allora $A[t/x] = B[t/x] \wedge C[t/x]$, $A[t/x] = B[t/x] \vee C[t/x]$, o $A[t/x] = B[t/x] \supset C[t/x]$, rispettivamente;
- se $A \equiv \forall y : r.B$, o $A \equiv \exists y : r.B$, e $y : r \equiv x : s$, allora $A[t/x] = A$;
- se $A \equiv \forall y : r.B$, o $A \equiv \exists y : r.B$, e $y : r \not\equiv x : s$, allora $A[t/x] = \forall z : r.(B[z/y])[t/x]$, o $A[t/x] = \exists z : r.(B[z/y])[t/x]$, rispettivamente, dove $z : r \notin \mathrm{FV}(B) \cup \mathrm{FV}(t)$.

Sostituzione

Le prime clausole della definizione sono ovvie: sostituiamo la variabile x con il termine t ove esso appaia.

La penultima clausola significa che una variabile legata non può essere sostituita: questo è semplice da comprendere, in quanto non ha senso sostituire x con 5 nella formula $\exists x : \mathbb{N}. x^2 = x^3$. Infatti, la formula è vera, in quanto $1^2 = 1 = 1^3$, ma, evidentemente, questo accade soltanto per **qualche** valore di x, che il quantificatore esistenziale ha il compito di identificare.

L'ultima clausola suona criptica. Essa dice che, prima di effettuare la sostituzione di x con t nella formula quantificata B, dobbiamo rinominare la variabile quantificata y con una **nuova** variabile, che non compaia in B e t.

Un esempio aiuta a chiarificare perché questo sia necessario: sia $A \equiv \exists x : \mathbb{N}. x + y = 2y$ e sia $t \equiv 2x$. Se non rinominiamo le variabili, $A[t/y]$ diverrebbe $\exists x : \mathbb{N}. x + 2x = 2(2x)$, ovvero, $\exists x : \mathbb{N}. 3x = 4x$. Notiamo che A vale ogniqualvolta $x = y$, ma questo è falso per $A[t/y]$. Il problema è che la x in t e quella in A dovrebbero essere tenute distinte—e noi otteniamo questo rinominando la variabile prima di effettuare la sostituzione.

Deduzione naturale

Fissato un linguaggio al prim'ordine, la definizione di *teoria* è la medesima già data per il caso proposizionale.

Lo stesso vale per la definizione di *prova* e gli altri concetti collegati, eccetto che la collezione delle regole d'inferenza contiene quattro nuove regole, deputate a gestire i quantificatori. Esse sono illustrate nelle slide che seguono.

Quando il linguaggio contiene l'uguaglianza, richiediamo la presenza di altre regole d'inferenza, dettagliate nelle slide seguenti.

La composizione modulare delle regole d'inferenza nella deduzione naturale spiega perché abbiamo scelto questo sistema rispetto ai molti altri equivalenti presenti in letteratura: tutti i sistemi di deduzione in questo corso sono ottenuti aggiungendo o togliendo alcune regole dal caso proposizionale o del prim'ordine.

Deduzione naturale

Seguendo la notazione già vista, le regole d'inferenza che governano la quantificazione universale sono

$$\frac{A}{\forall x : s.A}\,\forall\mathrm{I} \qquad \frac{\forall x : s.A}{A[t/x]}\,\forall\mathrm{E}$$

sotto le condizioni

- in $\forall E$, t è un termine di tipo s;
- in $\forall I$, la variabile $x : s$ non *occorre libera nella prova* dell'antecedente, il che significa che, per ogni assunzione G, $x : s \notin \mathrm{FV}(G)$. Questa condizione è talora riferita dicendo che $x : s$ è una *eigenvariable*.

Si noti la somiglianza tra le regole per il $\forall$ e quelle per $\wedge$.

Deduzione naturale

In modo simile, le regole per il quantificatore esistenziale sono

$$\frac{A[t/x]}{\exists x : s.\, A}\,\exists \mathsf{I} \qquad \frac{\exists x : s.\, B \quad \begin{array}{c}[B]\\ \vdots \\ A\end{array}}{A}\,\exists \mathsf{E}$$

sotto le condizioni

- in $\exists I$, t è un termine di tipo s;
- in $\exists E$, la variabile $x : s$ non compare libera nella prova del secondo antecedente, ovvero, per ogni assunzione G nella seconda sottoprova, eccetto che per B, $x : s \notin \mathrm{FV}(G)$ **e** $x : s \notin \mathrm{FV}(A)$. Di nuovo, $x : s$ è detta essere una eigenvariable. Si noti come questa regola d'inferenza scarichi l'assunzione B.

Si noti la somiglianza tra le regole per $\exists$ e quelle per $\vee$.

Deduzione naturale

L'uguaglianza è una relazione speciale, e questo fatto è catturato da una serie di regole di inferenza apposite. Quando il linguaggio contiene una relazione di uguaglianza per qualche sorta s, essa è soggetta alle seguenti regole:

$$\frac{}{\forall x : s.\, x = x}\ \text{refl} \qquad \frac{}{\forall x : s.\forall y : s.\, x = y \supset y = x}\ \text{sym}$$

$$\frac{}{\forall x : s.\forall y : s.\forall z : s.\, x = y \wedge y = z \supset x = z}\ \text{trans}$$

$$\frac{A[t/x] \quad t = r}{A[r/x]}\ \text{subst}$$

$$\frac{}{\forall x_1 : s_1 \ldots \forall x_n : s_n.\exists! z : s_0.\, z = f(x_1, \ldots, x_n)}\ \text{fun}$$

dove t e r sono termini di tipo s, e $f : s_1 \times \cdots \times s_n \to s_0$ è un simbolo di funzione nel linguaggio.

Riferimenti

Usualmente, la logica al prim'ordine è presentata in un modo semplificato, evitando di avere un linguaggio multi-sortato, e usando un numero ridotto di connettivi. Sebbene questo approccio semplifichi l'esposizione iniziale, esso rende difficile passare ad altri sistemi logici, ad esempio la logica intuizionista, e rende complesso trattare teorie matematiche e informatiche reali, dove sorte multiple sono spesso presenti.

Un buon testo che introduca il linguaggio al prim'ordine in modo formale è *John Bell* e *Moshé Machover*, A Course in Mathematical Logic, North-Holland, (1977).

La deduzione naturale è descritta in molti testi. Questa lezione segue *A.S. Troelstra* e *H. Schwichtenberg*, Basic Proof Theory, Cambridge Tracts in Theoretical Computer Science 43, Cambridge University Press, (1996).

Logica

Lezione 10

Dr Marco Benini

marco.benini@uninsubria.it

Dipartimento di Scienza e Alta Tecnologia
Università degli Studi dell'Insubria

a.a. 2020/21

Syllabus

Logica al prim'ordine:

- Esempi di deduzione
- Esempi di formalizzazione

Esempi

Esempio 10.1

$$\dfrac{\dfrac{\dfrac{\dfrac{[P]^1}{\exists x : s.\, P}\,\exists \mathsf{I} \qquad [\neg \exists x : s.\, P]^2}{\bot}\,\neg \mathsf{E}}{\neg P}\,\neg \mathsf{I}^1}{\forall x : s.\, \neg P}\,\forall \mathsf{I}}{(\neg \exists x : s.\, P) \supset \forall x : s.\, \neg P}\,\supset \mathsf{I}^2$$

Applicando la legge di doppia negazione ($A = \neg\neg A$), e ponendo $P \equiv \neg A$, otteniamo che $(\neg \exists x : s.\, \neg A) \supset \forall x : s.\, A$.

Esempi

Esempio 10.2

$$\dfrac{\dfrac{\dfrac{[\exists x : s.\, P]^1 \qquad \dfrac{\dfrac{[P]^2 \qquad \dfrac{[\forall x : s.\, \neg P]^3}{\neg P}\forall E}{\bot}\neg E}{\bot}\exists E^2}{\neg\exists x : s.\, P}\neg I^1}{(\forall x : s.\, \neg P) \supset \neg\exists x : s.\, P}\supset I^3$$

Ponendo $P \equiv \neg A$ e applicando la legge di doppia negazione si ottiene $\forall x : s.\, A = \neg\exists x : s.\, \neg A$, l'analogo al prim'ordine di una delle Leggi di De Morgan.

Esempi

Esempio 10.3

$$\dfrac{\dfrac{\dfrac{[\exists x : s.\, \neg P]^1 \qquad \dfrac{\dfrac{[\forall x : s.\, P]^2}{P}\,\forall \mathsf{E} \qquad [\neg P]^3}{\bot}\,\neg \mathsf{E}}{\bot}\,\exists \mathsf{E}^3}{\neg \forall x : s.\, P}\,\neg \mathsf{I}^2}{(\exists x : s.\, \neg P) \supset \neg \forall x : s.\, P}\,\supset \mathsf{I}^1$$

Esempi

Esempio 10.4

Per mostrare come la restrizione sulle variabili nella regola di introduzione del quantificatore universale sia essenziale, consideriamo il seguente controesempio. Sia $x : s \in \mathrm{FV}(P)$.

$$\dfrac{\dfrac{\dfrac{[P]^1}{\forall x : s.\, P}\,\forall \mathrm{I}}{P \supset \forall x : s.\, P}\,\supset \mathrm{I}^1}{\forall x : s.\, (P \supset \forall x : s.\, P)}\,\forall \mathrm{I}$$

L'istanza della regola $\forall I$ in cima è invalida, poiché $x : s$ appare nelle assunzioni che sono non scaricate in quel momento della prova.
In aritmetica, se P sta per 'x è pari', la conclusione permette di dimostrare che, dato che $P[0/x]$ è vero, ogni numero naturale è pari!

Esempi

Esempio 10.5

Un altro controesempio, che mostra come la restrizione sulle variabili sia essenziale nella regola di eliminazione del quantificatore esistenziale, è il seguente. Di nuovo, sia $x : s \in \mathrm{FV}(P)$.

$$\dfrac{\dfrac{\dfrac{\dfrac{[\exists x : s.\, P]^1 \qquad \dfrac{[P \supset Q]^2 \quad [P]^3}{Q}\supset\mathrm{E}}{Q}\exists\mathrm{E}^3}{(\exists x : s.\, P) \supset Q}\supset\mathrm{I}^1}{(P \supset Q) \supset ((\exists x : s.\, P) \supset Q)}\supset\mathrm{I}^2}{\forall x : s.((P \supset Q) \supset ((\exists x : s.\, P) \supset Q))}\forall\mathrm{I}$$

Nell'aritmetica, sia $Q \equiv \bot$, in modo che la conclusione si riduca a $\forall x : s.(\neg P \supset \neg \exists x : s.\, P)$. Se P sta per 'x è pari', poiché $P[1/x]$ è falso, la conclusione permette di dedurre che non vi sono numeri naturali pari!

Esempi

Esempio 10.6

L'ultimo controesempio mostra il motivo della restrizione che la variabile quantificata non debba comparire libera nella conclusione della regola di eliminiazione di esiste.
Sia $x \in \mathrm{FV}(A)$:

$$\dfrac{\dfrac{\dfrac{[\exists x : s. A]^1 \quad [A]^2}{A}\,\exists \mathrm{E}^2}{\forall x : s. A}\,\forall \mathrm{I}}{(\exists x : s. A) \supset (\forall x : s. A)}\,\supset \mathrm{I}^1$$

Nell'aritmetica, sia A la formula che afferma che il suo argomento è pari. Poiché c'è almeno un numero pari, 2 per esempio, ne segue che ogni numero debba essere pari.

Esempi

Esempio 10.7

$\vdash \forall x.B \supset A = B \supset \forall x.A$ con $x \notin \mathrm{FV}(B)$

$$\dfrac{\dfrac{\dfrac{\dfrac{[B]^1 \qquad \dfrac{[\forall x.B \supset A]^2}{B \supset A}\,\forall\mathrm{E}}{A}\,\supset\mathrm{E}}{\forall x.A}\,\forall\mathrm{I}}{B \supset \forall x.A}\,\supset\mathrm{I}^1}{(\forall x.B \supset A) \supset (B \supset \forall x.A)}\,\supset\mathrm{I}^2 \qquad \dfrac{\dfrac{\dfrac{\dfrac{[B \supset \forall x.A]^1 \quad [B]^2}{\forall x.A}\,\supset\mathrm{E}}{A}\,\forall\mathrm{E}}{B \supset A}\,\supset\mathrm{I}^2}{\forall x.B \supset A}\,\forall\mathrm{I}}{(B \supset \forall x.A) \supset (\forall x.B \supset A)}\,\supset\mathrm{I}^1$$

Esempi

Esempio 10.8

$\vdash \forall x.A \supset B = (\exists x.A) \supset B$ con $x \notin \mathrm{FV}(B)$

$$\dfrac{\dfrac{[\exists x.A]^1 \qquad \dfrac{\dfrac{[A]^2 \qquad \dfrac{[\forall x.A \supset B]^3}{A \supset B}\forall E}{B}\supset E}{B}\exists E^2}{(\exists x.A) \supset B}\supset I^1}{(\forall x.A \supset B) \supset ((\exists x.A) \supset B)}\supset I^3$$

$$\dfrac{\dfrac{\dfrac{[A]^1}{\exists x.A}\exists I \qquad [(\exists x.A) \supset B]^2}{B}\supset E}{\dfrac{A \supset B}{\forall x.A \supset B}\forall I}\supset I^1}{((\exists x.A) \supset B) \supset (\forall x.A \supset B)}\supset I^2$$

Esempi

Esempio 10.9

$\vdash \neg\neg\forall x.A \supset \forall x.\neg\neg A$

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{[\forall x.A]^1}{A}\forall E \quad [\neg A]^2}{\bot}\neg E}{\neg\forall x.A}\neg I^1 \quad [\neg\neg\forall x.A]^3}{\bot}\neg E}{\neg\neg A}\neg I^2}{\forall x.\neg\neg A}\forall I}{\neg\neg(\forall x.A) \supset \forall x.\neg\neg A}\supset I^3$$

Esempi

Esempio 10.10

$\vdash A \wedge (\exists x.\, B) \supset \exists x.\, A \wedge B$ con $x \notin \mathrm{FV}(A)$

$$\dfrac{\dfrac{[A \wedge \exists x.\, B]^1}{\exists x.\, B}\,\wedge\mathsf{E}_2 \qquad \dfrac{\dfrac{[A \wedge \exists x.\, B]^1}{A}\,\wedge\mathsf{E}_1 \qquad \dfrac{[B]^2}{A \wedge B}\,\wedge\mathsf{I}}{\exists x.\, A \wedge B}\,\exists\mathsf{I}}{\dfrac{\exists x.\, A \wedge B}{A \wedge (\exists x.\, B) \supset \exists x.\, A \wedge B}\,\supset\mathsf{I}^1}\,\exists\mathsf{E}^2$$

Esempi

Esempio 10.11

$\vdash \exists x.A \wedge B \supset A \wedge \exists x.B$ con $x \notin \mathrm{FV}(A)$

$$\dfrac{\dfrac{[\exists x.A \wedge B]^1 \qquad \dfrac{\dfrac{\dfrac{[A \wedge B]^2}{A}\wedge \mathrm{E}_1 \qquad \dfrac{\dfrac{[A \wedge B]^2}{B}\wedge \mathrm{E}_2}{\exists x.B}\exists \mathrm{I}}{A \wedge \exists x.B}\wedge \mathrm{I}}{A \wedge \exists x.B}\exists \mathrm{E}^2}{(\exists x.A \wedge B) \supset A \wedge \exists x.B}\supset \mathrm{I}^1$$

Esempi

Esempio 10.12

$\vdash A \wedge \forall x.B = \forall x.A \wedge B$ con $x \notin \mathrm{FV}(A)$

$$\dfrac{\dfrac{\dfrac{\dfrac{[A \wedge \forall x.B]^1}{A}\wedge\mathrm{E}_1 \quad \dfrac{\dfrac{[A \wedge \forall x.B]^1}{\forall x.B}\wedge\mathrm{E}_2}{B}\forall\mathrm{E}}{A \wedge B}\wedge\mathrm{I}}{\forall x.A \wedge B}\forall\mathrm{I}}{A \wedge (\forall x.B) \supset \forall x.A \wedge B}\supset\mathrm{I}^1$$

$$\dfrac{\dfrac{\dfrac{\dfrac{[\forall x.A \wedge B]^1}{A \wedge B}\forall\mathrm{E}}{A}\wedge\mathrm{E}_1 \quad \dfrac{\dfrac{[\forall x.A \wedge B]^1}{A \wedge B}\forall\mathrm{E}}{B}\wedge\mathrm{E}_2}{A \wedge \forall x.B}\wedge\mathrm{I}}{(\forall x.A \wedge B) \supset A \wedge \forall x.B}\supset\mathrm{I}^1$$

Esempi

Esempio 10.13

$\vdash (\forall x.A \wedge B) \supset (\forall x.A) \wedge (\forall x.B)$

$$\dfrac{\dfrac{\dfrac{\dfrac{[\forall x.A \wedge B]^1}{A \wedge B}\wedge E_1}{A}\forall I}{\forall x.A} \quad \dfrac{\dfrac{\dfrac{[\forall x.A \wedge B]^1}{A \wedge B}\wedge E_2}{B}\forall I}{\forall x.B}}{(\forall x.A) \wedge (\forall x.B)}\wedge I}{(\forall x.A \wedge B) \supset (\forall x.A) \wedge (\forall x.B)}\supset I^1$$

Esempi

Esempio 10.14

$\vdash (\forall x.A) \wedge (\forall x.B) \supset \forall x.A \wedge B$

$$\dfrac{\dfrac{\dfrac{\dfrac{[(\forall x.A) \wedge (\forall x.B)]^1}{\forall x.A}\wedge E_1}{A}\forall E \quad \dfrac{\dfrac{[(\forall x.A) \wedge (\forall x.B)]^1}{\forall x.B}\wedge E_2}{B}\forall E}{A \wedge B}\wedge I}{\forall x.A \wedge B}\forall I}{(\forall x.A) \wedge (\forall x.B) \supset \forall x.A \wedge B}\supset I^1$$

Esempi

Esempio 10.15

$\vdash (\exists x.A \wedge B) \supset (\exists x.A) \wedge (\exists x.B)$

$$\dfrac{\dfrac{[\exists x.A \wedge B]^1 \qquad \dfrac{\dfrac{\dfrac{[A \wedge B]^2}{A}\wedge E_1}{\exists x.A}\exists I \qquad \dfrac{\dfrac{[A \wedge B]^2}{B}\wedge E_2}{\exists x.B}\exists I}{(\exists x.A) \wedge (\exists x.B)}\wedge I}{(\exists x.A) \wedge (\exists x.B)}\exists E^2}{(\exists x.A \wedge B) \supset (\exists x.A) \wedge (\exists x.B)}\supset I^1$$

Esempi

Esempio 10.16

$\vdash (\exists x.\forall y.A) \supset \forall y.\exists x.A$

$$\dfrac{\dfrac{[\exists x.\forall y.A]^1 \quad \dfrac{\dfrac{\dfrac{[\forall y.A]^2}{A}\,\forall E}{\exists x.A}\,\exists I}{\forall y.\exists x.A}\,\forall I}{\forall y.\exists x.A}\,\exists E^2}{(\exists x.\forall y.A) \supset \forall y.\exists x.A}\,\supset I^1$$

Esempi

Esempio 10.17

In aritmetica, si esprima la proprietà P che ogni numero possa essere diviso per 2 con resto 0 oppure 1.

1. "ogni numero": $P = \forall x.\square$;
2. "può essere diviso per 2 con un resto": $x = 2y + z$;
3. quindi $P = \forall x.\exists y.\exists z.x = 2y + z \wedge \square$;
4. il resto può essere 0 oppure 1: $z = 0 \vee z = 1$;
5. quindi $P = \forall x.\exists y.\exists z.x = 2y + z \wedge (z = 0 \vee z = 1)$.

Esempi

Esempio 10.18

Si esprima che vi siano al massimo due valori y per ogni x per i quali la proprietà P valga.

1. "ci sono due valori": $\exists v_1.\exists v_2.\Box$;
2. "ci sono al più due valori y". $\forall y.y = v_1 \vee y = v_2$;
3. quindi $\forall x.\exists v_1.\exists v_2.\forall y.P \supset y = v_1 \vee y = v_2$. esprime la richiesta;
4. si noti come P possa non valere (l'implicazione è vera);
5. essa può valere per soltanto un valore (allora $v_1 = v_2$);
6. essa può valere per due valori distinti;
7. ma non può valere per tre o più valori.

Esempi

Esempio 10.19

Si sostituisca la regola di "introduzione di per ogni" del calcolo naturale con la seguente regola con infinite premesse:

$$\frac{\left\{\begin{array}{c}\Gamma\\ \vdots\\ A[t/x]\end{array}\right\}_{t \text{ termine}}}{\forall x.A}\ \forall \mathrm{I}^{\infty}$$

Mostrare che ogni formula che possa essere dedotta in questo calcolo, può anche essere dedotta nel calcolo in deduzione naturale.

In ogni prova π in cui la regola $\forall \mathrm{I}^{\infty}$ occorre, soltanto un numero finito di assunzioni in Γ è utilizzato. Quindi, in particolare, solo un numero finito di variabili occorre in queste assunzioni. Si prenda una variabile z che non sia tra queste variabili. Allora esiste una prova $\theta : \Gamma \vdash A[z/x]$ tra le premesse di $\forall \mathrm{I}^{\infty}$. Pertanto, nel calcolo in deduzione naturale, θ può essere usata per dedurre $\forall z.A[z/x]$ per $\forall \mathrm{I}$, che equivale a $\forall x.A$ rinominando la variabile legata.

Esempi

Esempio 10.20

Si mostri che ogni formula che possa essere dedotta in deduzione naturale, può essere dedotta nel calcolo con la regola $\forall \mathrm{I}^{\infty}$.

Si consideri

$$\dfrac{\begin{array}{c}\Gamma \\ \vdots \\ A\end{array}}{\forall x.A}\,\forall \mathrm{I}$$

con x che non occorra libera nelle assunzioni. Per una semplice induzione sulla struttura delle prove, si può dimostrare che, se $\pi\colon \Delta \vdash B$ allora $\pi[t/z]\colon \Delta[t/z] \vdash B[t/z]$ è una dimostrazione. Sostituendo t per x nella premessa di $\forall \mathrm{I}$, per tutti i termini t della stessa sorta di x, vediamo che tutte le premesse di $\forall \mathrm{I}^{\infty}$ sono valide, così come la sua conclusione.

Riferimenti

Esercizi possono essere reperiti in ogni libro di testo di logica, ad esempio nel Capitolo 1 di *John Bell* e *Moshé Machover*, A Course in Mathematical Logic, North-Holland, (1977).

Altri esercizi svolti sono disponibili sulla pagina web del corso.

Logica

Lezione 11

Dr Marco Benini

marco.benini@uninsubria.it

Dipartimento di Scienza e Alta Tecnologia
Università degli Studi dell'Insubria

a.a. 2020/21

Syllabus

Logica al prim'ordine:

- Significato inteso
- Semantica di Tarski
- Teorema di correttezza

Significato inteso

Fissata una segnatura $\langle S; F; R \rangle$, l'interpretazione intesa di una sorta $s \in S$ è uno specifico insieme; l'interpretazione intesa di un simbolo di funzione è una funzione; e l'interpretazione intesa di un simbolo di relazione è una relazione.

Il significato inteso dell'uguaglianza, $=: s \times s$, quando presente nel linguaggio, è l'identità dei suoi argomenti.

Pertanto, il significato inteso di un termine è un elemento che è identificato attraverso l'interpretazione delle funzioni e la valutazione delle variabili.

Significato inteso

A loro volta, le formule stanno per un valore di verità, o vero o falso, come nel caso proposizionale. E i connettivi hanno il significato inteso proposizionale, che abbiamo già illustrato.

Le formule atomiche, $r(t_1, \ldots, t_n)$, sono vere esattamente quando l'argomento $(t_1, \ldots, t_n)$ appartiene alla relazione denotata da r.

Una formula è universalmente valida, ovvero $\forall x : s. A$ è vera, quando A è vera in qualsiasi modo possiamo interpretare x come un elemento dell'insieme denotato da s.

Simmetricamente, una formula è esistenzialmente valida, ovvero $\exists x : s. A$ è vera, quando esiste un elemento e nell'insieme denotato da s tale che interpretando x come e renda A vera.

Semantica

La semantica standard per la logica al prim'ordine, sviluppata Alfred Tarski, formalizza direttamente l'interpretazione intesa.

Definizione 11.1 (Σ-struttura)

Sia $\Sigma = \langle S; F, R\rangle$ una segnatura.
Allora, una Σ-*struttura* $\mathscr{M} = \langle U; \mathscr{F}; \mathscr{R}\rangle$ è composta da

- una collezione $U = \{u_s\}_{s\in S}$ di insiemi non vuoti, detta l'*universo*,
- una collezione di funzioni sull'universo
  $\mathscr{F} = \{g_f : u_{s_1} \times \cdots \times u_{s_n} \to u_{s_0} \mid f : s_1 \times \cdots \times s_n \to s_0 \in F\}$,
- una collezione di relazioni sull'universo
  $\mathscr{R} = \{\rho_r : u_{s_1} \times \cdots \times u_{s_n} \mid r : s_1 \times \cdots \times s_n \in R\}$.

Semantica

Per rendere chiara la relazione tra una segnatura e una Σ-struttura, adottiamo la seguente notazione:

- per ogni $s \in S$, $[\![s]\!] = u_s$;
- per ogni $f : s_1 \times \cdots \times s_n \to s_0 \in F$, $[\![f]\!] = g_f$;
- per ogni $r : s_1 \times \cdots \times s_n \in R$, $[\![r]\!] = \rho_r$.

Questa è chiamata l'*interpretazione della segnatura* nella Σ-struttura.

Semantica

Definizione 11.2 (Interpretazione dei termini)

Sia $\Sigma = \langle S; F, R \rangle$ una segnatura, e sia $\mathscr{M}$ una Σ-struttura, con la notazione come prima. Inoltre, sia $\nu = \{\nu_s\}_{s \in S}$ una collezione di funzioni $\nu_s \colon \{v \colon v \colon s \in V\} \to [\![s]\!]$, che mappano le variabili di tipo s nel corrispondente insieme $[\![s]\!]$.
Quindi, un termine t viene interpretato secondo la seguente definizione induttiva sulla sua struttura:

- se $t \in V$ è una variabile di tipo s, allora $[\![t]\!] = \nu_s(t)$;
- se $t \equiv f(t_1, \ldots, t_n)$, allora $[\![t]\!] = [\![f]\!]([\![t_1]\!], \ldots, [\![t_n]\!])$.

Definizione 11.3 (Interpretazione delle formule)

Sia $\Sigma = \langle S; F, R\rangle$ una segnatura, sia $\mathscr{M}$ una Σ-struttura, e sia ν una *valutazione delle variabili*, con la notazione come in precedenza.
Quindi, una formula A è interpretata secondo la seguente definizione induttiva sulla sua struttura:

- se $A \equiv \top$, $[\![A]\!] = 1$;
- se $A \equiv \bot$, $[\![A]\!] = 0$;
- se $A \equiv r(t_1, \ldots, t_n)$, $[\![A]\!] = 1$ quando $([\![t_1]\!], \ldots, [\![t_n]\!]) \in [\![r]\!]$, e $[\![A]\!] = 0$ altrimenti;
- se $A \equiv \neg B$, $A \equiv B \wedge C$, $A \equiv B \vee C$, $A \equiv B \supset C$ allora $[\![A]\!]$ è definita come nella semantica a tavole di verità;
- se $A \equiv \forall x : s.\, B$ o $A \equiv \exists x : s.\, B$, sia $\xi = \{\xi_s\}_{s \in S}$ una valutazione di variabili tale che $\xi_\alpha = \nu_\alpha$ per ogni $\alpha \neq s$, e $\xi_s(v) = \nu_s(v)$ per ogni $v \neq x$. Allora, $[\![\forall x : s.\, B]\!] = 1$ se, per ogni possibile ξ, $[\![B]\!] = 1$, e $[\![\forall x : s.\, B]\!] = 0$ altrimenti. Inoltre, $[\![\exists x : s.\, B]\!] = 1$ se esiste uno ξ tale che $[\![B]\!] = 1$, e $[\![\exists x : s.\, B]\!] = 0$ altrimenti.

Semantica

Stipuliamo che, se l'uguaglianza è presente nel linguaggio, $[\![t_1 = t_2]\!] = 1$ esattamente quando $[\![t_1]\!] = [\![t_2]\!]$.

In altri termini, $[\![=_s]\!]$, l'uguaglianza sulla sorta s, rappresenta la *relazione diagonale* $\{(x,x) : x \in [\![s]\!]\}$.

Vale la pena sottolineare che l'uguaglianza è sempre tipata: $t_1 = t_2$ è una formula se e solo se t_1 e t_2 sono termini della stessa sorta s, e la relazione $=$ deve essere letta come una abbreviazione per $=_s$, che sta per la relazione diagonale sull'insieme denotato dalla sorta s.

Correttezza

Definizione 11.4 (Validità)

Una formula A è *valida* o *vera* in una Σ-struttura $\mathscr{M}$ assieme ad una interpretazione v delle variabili, quando $[\![A]\!] = 1$.
Un insieme di formule è *valido* o *vero* quando ogni formula nell'insieme è valida. La coppia $(\mathscr{M}, v)$ è un *modello* per la teoria T quando essa rende vera ogni formula in T.

Correttezza

Teorema 11.5 (Correttezza)

In ogni modello $(\mathscr{M}, v)$ della teoria T, che rende vere le assunzioni nell'insieme finito Δ, se $\pi : \Delta \vdash_T A$ allora A è valida.

Dimostrazione. (i)

Iniziamo osservando che, per la Definizione 11.3, i connettivi agiscono nell'algebra Booleana su $\{0,1\}$ con $0 < 1$, quindi le operazioni $\wedge$, $\vee$ $\neg$ sono definite come nella semantica a tavole di verità.

La prova è per induzione sulla struttura della dimostrazione π: mostriamo che l'interpretazione della conclusione A è 1 quando l'interpretazione di ogni G nell'insieme finito di assunzioni Γ è 1: ↪

Correttezza

↪ Dimostrazione. (ii)

- se π è una prova per assunzione, allora $A \in \Gamma$ e, per ipotesi $[\![A]\!] = 1$.
- se π è una prova per assioma, allora $A \in T$, e, per ipotesi, $[\![A]\!] = 1$.
- se π è una istanza della Legge del Terzo Escluso, allora $A \equiv B \vee \neg B$, e $[\![A]\!] = [\![B \vee \neg B]\!] = [\![B]\!] \vee \neg [\![B]\!] = 1$ per definizione di complemento.
- se π è una istanza di introduzione di $\top$, allora $A \equiv \top$, quindi $[\![A]\!] = 1$.
- se π è una istanza di refl, allora $A \equiv \forall x : s.x = x$, quindi $[\![A]\!] = 1$ quando $[\![x = x]\!] = 1$ per ogni possibile valutazione della variabile x in $[\![s]\!]$. Pertanto, se x viene mappata in $e \in [\![s]\!]$, $(e, e) \in \{(z, z) : z \in [\![s]\!]\}$, quindi $[\![x = x]\!] = 1$ per qualsiasi e.
- se π è un'istanza di sym, allora $A \equiv \forall x : s.\forall y : s.x = y \supset y = x$, quindi $[\![A]\!] = 1$ quando $[\![x = y \supset y = x]\!] = 1$ per ogni possibile valutazione delle variabili x e y in $[\![s]\!]$. Pertanto, se x viene mappata in $e_x \in [\![s]\!]$, e y in $e_y \in [\![s]\!]$, se $(e_x, e_y) \in \{(z, z) : z \in [\![s]\!]\}$, allora $e_x = e_y$, perciò $(e_y, e_x) \in \{(z, z) : z \in [\![s]\!]\}$, ovvero $[\![x = y \supset y = x]\!] = 1$. ↪

Correttezza

↪ Dimostrazione. (iii)

- se π è un'istanza di trans, allora

$$A \equiv \forall x : s. \forall y : s. \forall z : s. x = y \wedge y = z \supset x = z \ ,$$

quindi $[\![A]\!] = 1$ quando $[\![x = y \wedge y = z \supset x = z]\!] = 1$ per ogni possibile valutazione delle variabili x, y e z in $[\![s]\!]$. Quindi, se x viene mappata in $e_x \in [\![s]\!]$, y in $e_y \in [\![s]\!]$ e z in $e_z \in [\![s]\!]$, se $(e_x, e_y) \in \{(z,z) : z \in [\![s]\!]\}$ e $(e_y, e_z) \in \{(z,z) : z \in [\![s]\!]\}$, allora $e_x = e_y = e_z$, e quindi $(e_x, e_z) \in \{(z,z) : z \in [\![s]\!]\}$, ovvero, $[\![x = y \wedge y = z \supset x = z]\!] = 1$.

- se π è un'istanza di fun, allora

$$A \equiv \forall x_1 : s_1 \ldots \forall x_n : s_n. \exists! z : s_0. z = f(x_1, \ldots, x_n) \ ,$$

quindi $[\![A]\!] = 1$ esattamente quando z possa essere univocamente mappata in un valore e_z in $[\![s_0]\!]$ in modo tale che $(e_z, [\![f]\!](e_{x_1}, \ldots, e_{x_n})) \in \{(z,z) : z \in [\![s]\!]\}$, che è evidentemente vero per $e_z = [\![f]\!](e_{x_1}, \ldots, e_{x_n})$. ↪

Correttezza

↪ Dimostrazione. (iv)

- se π è un'istanza di subst, allora, per ipotesi induttiva, $[\![A[t/x]]\!] = 1$ e $[\![t = r]\!] = 1$, ovvero $[\![t]\!] = [\![r]\!]$. La conclusione segue da una semplice induzione sulla struttura della formula A.
- se π è un'istanza di eliminazione di $\bot$, allora, per ipotesi induttiva, $0 = [\![\bot]\!] = 1$. Quindi, $[\![A]\!] = 1$ poiché l'interpretazione è una funzione.
- se π è un'istanza di introduzione di $\wedge$, allora $A \equiv B \wedge C$, e per ipotesi induttiva due volte, $[\![B]\!] = 1$ e $[\![C]\!] = 1$. Quindi, $1 = [\![B]\!] \wedge [\![C]\!] = [\![A]\!]$.
- se π è un'istanza di eliminazione di $\wedge$ a sinistra, allora, per ipotesi induttiva, per qualche formula B, $[\![A \wedge B]\!] = [\![A]\!] \wedge [\![B]\!] = 1$. Quindi, per definizione di $\wedge$, $[\![A]\!] = 1$.
- se π è un'istanza di eliminazione di $\wedge$ a destra, allora, per ipotesi induttiva, per qualche formula B, $[\![B \wedge A]\!] = [\![B]\!] \wedge [\![A]\!] = 1$. Quindi, per definizione di $\wedge$, $[\![A]\!] = 1$. ↪

Correttezza

↪ Dimostrazione. (v)

- se π è un'istanza di introduzione di $\vee$ a sinistra, allora $A \equiv B \vee C$ e, per ipotesi induttiva, $[\![B]\!] = 1$. Quindi, per definizione di $\vee$, $1 = [\![B]\!] \vee [\![C]\!] = [\![A]\!]$.
- se π è un'istanza di introduzione di $\vee$ a destra, allora $A \equiv B \vee C$ e, per ipotesi induttiva, $[\![C]\!] = 1$. Quindi, per definizione di $\vee$, $1 = [\![B]\!] \vee [\![C]\!] = [\![A]\!]$.
- se π è un'istanza di eliminazione di $\vee$, allora, per ipotesi induttiva, per qualche formula B e C, $[\![B \vee C]\!] = [\![B]\!] \vee [\![C]\!] = 1$, se $[\![B]\!] = 1$ allora $[\![A]\!] = 1$, e se $[\![C]\!] = 1$ allora $[\![A]\!] = 1$. Per definizione di $\vee$, o $[\![B]\!] = 1$ e quindi $[\![A]\!] = 1$, oppure $[\![C]\!] = 1$ e quindi $[\![A]\!] = 1$. ↪

Correttezza

↪ Dimostrazione. (vi)

- se π è un'istanza di introduzione di $\supset$, allora $A \equiv B \supset C$ per qualche formula B e C. Per ipotesi induttiva, se $[\![B]\!] = 1$ allora $[\![C]\!] = 1$. Quindi, per definizione di $\supset$, $[\![A]\!] = 1$.
- se π è un'istanza di eliminazione di $\supset$, allora, per qualche formula B e per ipotesi induttiva due volte, $[\![B \supset A]\!] = 1$ e $[\![B]\!] = 1$. Per definizione di $\supset$, $[\![A]\!] = 1$.
- se π è un'istanza di introduzione di $\neg$, allora $A \equiv \neg B$ per qualche formula B. Quindi, per ipotesi induttiva, se $[\![B]\!] = 1$ allora $0 = [\![\bot]\!] = 1$. Quindi, $[\![\neg B]\!] = 1$ poiché, o $[\![B]\!] = 0$, oppure $0 = 1$.
- se π è un'istanza di eliminazione di $\neg$, allora $A \equiv \bot$ e, per ipotesi induttiva due volte, $[\![\neg B]\!] = 1$ e $[\![B]\!] = 1$. Quindi, per definizione di complemento, $0 = 1$. Perciò $0 = [\![A]\!] = 1$. ↪

Correttezza

↪ Dimostrazione. (vii)

- se π è un'istanza di introduzione di $\forall$, allora $A \equiv \forall x\colon s.B$, e, per ipotesi induttiva, $[\![B]\!] = 1$ per ogni valutazione di variabili che renda vere tutte le assunzioni. Ma, dato che $x\colon s$ non compare libera in alcuna assunzione, $[\![B]\!] = 1$ in ogni modo in cui possiamo valutare x in $[\![s]\!]$, ovvero $[\![A]\!] = 1$.
- se π è un'istanza di eliminazione di $\forall$, allora $A \equiv B[t/x]$, e, per ipotesi induttiva, $[\![\forall x\colon s.B]\!] = 1$. Quindi, in particolare, quando x viene valutata in $[\![t]\!]$, $[\![A]\!] = [\![B[t/x]]\!] = 1$. ↪

Correttezza

↪ Dimostrazione. (viii)

- se π è un'istanza di introduzione di $\exists$, allora $A \equiv \exists x : s.B$, e, per ipotesi induttiva, $[\![B[t/x]]\!] = 1$. Quindi, la valutazione di variabili ξ_s che è identica a ν_s eccetto che per $\xi_s(x) = [\![t]\!]$ rende A valida.
- se π è un'istanza di eliminazione di $\exists$, allora, per ipotesi induttiva, $[\![\exists x : s.B]\!] = 1$ e, se $[\![B]\!] = 1$, allora A è vera. Ma, $[\![\exists x : s.B]\!] = 1$ significa che esiste un modo per valutare x in $[\![s]\!]$ che rende B valida. Applicando questa valutazione di variabili alla seconda ipotesi induttiva, otteniamo che A è valida. □

Riferimenti

L'interpretazione delle formula, come illustrata in questa lezione, è stata formalizzata per la prima volta da Alfred Tarski. Questa è una definizione classica, e può essere trovata nella maggior parte dei libri di testo.

Il Teorema di Correttezza è un risultato classico e la sua prova può essere trovata nella maggior parte dei testi. La nostra presentazione segue il già citato *John Bell* e *Moshé Machover*, A Course in Mathematical Logic, North-Holland, (1977). Vale la pena confrontare la dimostrazione data in questa lezione con la prova proposizionale che usa la semantica a tavole di verità.

Logica

Lezione 12

Dr Marco Benini

marco.benini@uninsubria.it

Dipartimento di Scienza e Alta Tecnologia
Università degli Studi dell'Insubria

a.a. 2020/21

Syllabus

Logica al prim'ordine:

- Risoluzione
- Unificazione

Risoluzione proposizionale

Ricordiamo che una *clausola* in logica proposizionale, è una disgiunzione finita di zero o più *letterali*. Un letterale è una variabile oppure una variabile negata. La clausola vuota, con zero letterali, è il falso, $\bot$.

Ricordiamo che ogni formula proposizionale può essere espressa in *forma normale congiuntiva*, ovvero come la congiunzione di clausole.

È più semplice e conveniente rappresentare una clausola come l'insieme dei suoi letterali. Allo stesso modo conviene pensare ad una forma normale congiuntiva come ad un insieme di clausole.

Risoluzione proposizionale

Definizione 12.1 (Risoluzione)

Siano α e β due clausole tali che, per un qualche letterale l, $l \in \alpha$ e $\neg l \in \beta$. Allora si dice *risolvente* la clausola $(\alpha \setminus \{l\}) \cup (\beta \setminus \{\neg l\})$.

In forma di regola d'inferenza:

$$\frac{A \cup \{l\} \quad B \cup \{\neg l\}}{A \cup B}\,\text{res}$$

Risoluzione proposizionale

Definizione 12.2 (Dimostrazione)

Sia S un insieme di clausole. Una *dimostrazione* per risoluzione della clausola R da S è una sequenza finita $C_1, \ldots, C_n$ tale che

- $C_n = R$
- per ogni $1 \leq i \leq n$, $C_i \in S$ oppure C_i è il risolvente di C_j, C_k con $j < i$ e $k < i$.

Scriveremo $S \vdash_{\text{res}} R$ per indicare una derivazione di R da S.

Una dimostrazione $S \vdash_{\text{res}} \bot$ è detta *refutazione*.

Risoluzione proposizionale

Enunciato 12.3

Siano A e B due clausole ed R un loro risolvente. Se A e B sono vere, allora R è valida.

Dimostrazione.

Se R è ottenuto per risoluzione da A e B, allora $A = A' \cup \{l\}$, $B = B' \cup \{\neg l\}$ e $R = A' \cup B'$.

Sia ν una valutazione di variabili e $[\![\ ]\!]$ la corrispondente interpretazione:

- se $[\![l]\!] = 1$, allora necessariamente $[\![B']\!] = 1$, e quindi $[\![R]\!] = 1$;
- altrimenti $[\![l]\!] = 0$, allora necessariamente $[\![A']\!] = 1$, e quindi $[\![R]\!] = 1$. □

Risoluzione proposizionale

Teorema 12.4

Un insieme finito di clausole S è insoddisfacibile se e solo se $S \vdash_{\mathsf{res}} \bot$.

Dimostrazione. (i)

Sia $S \vdash_{\mathsf{res}} \bot$. Supponiamo che S sia soddisfacibile.
Quindi esiste una interpretazione $[\![\ ]\!]$ tale che $[\![S]\!] = 1$. Per induzione sulla dimostrazione e per l'Enunciato 12.3, è immediato concludere che $[\![\bot]\!] = 1$, impossibile. Pertanto, S deve essere insoddisfacibile. ↪

Risoluzione proposizionale

↪ Dimostrazione. (ii)

Sia S insoddisfacibile.
Per induzione su n, dimostriamo che se T è insoddisfacibile e contiene n variabili distinte, allora $T \vdash_{\text{res}} \bot$:

- se $n = 0$, allora necessariamente $T = \{\bot\}$, in quanto $T = \emptyset$ è soddisfacibile da qualsiasi assegnamento. Quindi $T \vdash_{\text{res}} \bot$ in modo banale.
- se T contiene $n+1$ variabili, possiamo supporre senza ledere la generalità che nessuna clausola di T contenga un letterale e la sua negazione. Prendiamo una variabile x e costruiamo

$$
\begin{aligned}
T_x &= \{c \setminus \{x\} : c \in T \wedge \neg x \notin c\} \\
T'_x &= \{c : c \in T \wedge \neg x \notin c\} \\
T_{\neg x} &= \{c \setminus \{\neg x\} : c \in T \wedge x \notin c\} \\
T'_{\neg x} &= \{c : c \in T \wedge x \notin c\}
\end{aligned}
$$

T_x e $T_{\neg x}$ sono insoddisfacibili e contengono n variabili. ↪

Risoluzione proposizionale

↪ Dimostrazione. (iii)

Infatti, se T_x fosse soddisfatto da σ, allora σ estesa ad x ponendo $\sigma(x) = 0$ renderebbe vero T contro ipotesi. Analogamente per $T_{\neg x}$.
Quindi, per ipotesi induttiva $T_x \vdash_{\text{res}} \bot$ e $T_{\neg x} \vdash_{\text{res}} \bot$.
Nelle due dimostrazioni, ripristinando la variabile x ove sia stata elisa, otteniamo che $T'_x \vdash_{\text{res}} \bot$ oppure $T'_x \vdash_{\text{res}} x$, e $T'_{\neg x} \vdash_{\text{res}} \bot$ oppure $T'_{\neg x} \vdash_{\text{res}} \neg x$.

Se $T'_x \vdash_{\text{res}} \bot$ o $T'_{\neg x} \vdash_{\text{res}} \bot$, allora $T \vdash_{\text{res}} \bot$, e abbiamo concluso.

Altrimenti $T'_x \vdash_{\text{res}} x$ e $T'_{\neg x} \vdash_{\text{res}} \neg x$, quindi $T \vdash_{\text{res}} x$ e $T \vdash_{\text{res}} \neg x$. Quindi, con un passo di risoluzione, $T \vdash_{\text{res}} \bot$. □

Unificazione

Fissiamo un linguaggio al prim'ordine.

Definizione 12.5 (Unificatore)

Una sostituzione si dice *unificatore* di un insieme di termini $T = \{t_1, \ldots, t_n\}$ se $t_1\sigma = \cdots = t_n\sigma$.
L'insieme T è detto *unificabile* se esiste un unificatore per esso.

Unificazione

Definizione 12.6

Una sostituzione σ è *più generale* della sostituzione θ se esiste una sostituzione ν tale che $\theta = \sigma \circ \nu$.

Definizione 12.7

Un unificatore σ per un insieme T è detto *unificatore più generale* se, per ogni altro unificatore θ di T, σ è più generale di θ.

Unificazione

Teorema 12.8

Se un insieme T è unificabile, allora T ha un unificatore più generale unico a meno di ridenominazione delle variabili.

La dimostrazione, che omettiamo, consiste nel dimostrare la correttezza e la terminazione di un semplice algoritmo che decide se T è unificabile, e, nel caso, costruisce l'unificatore più generale.
Sebbene la dimostrazione non sia complessa, essa è alquanto tecnica e non essenziale, soprattutto considerando che gli algoritmi di unificazione effettivamente in uso sono più sofisticati per aumentarne l'efficienza.

Risoluzione al primo ordine

La regola d'inferenza di risoluzione può essere estesa alla logica al prim'ordine. Invece di avere variabili proposizionali, consideriamo letterali formati da formule atomiche e da formule atomiche negate.

Chiaramente, la presenza di variabili per oggetti di un universo invece che per valori di verità richiede un trattamento differente, che coinvolge l'unificazione.

Risoluzione al primo ordine

Definizione 12.9 (Risoluzione)

Siano A e B due clausole tali che $\mathrm{FV}(A) \cap \mathrm{FV}(B) = \emptyset$. Se esistono due insiemi di letterali $\{a_1, \ldots, a_m\} \subseteq A$ con $m \geq 1$ e $\{b_1, \ldots, b_k\} \subseteq B$ con $k \geq 1$, tali che $L = \{a_1, \ldots, a_m, b_1, \ldots, b_k\}$ sia unificabile, allora la clausola

$$R = ((A \setminus \{a_1, \ldots, a_m\}) \cup (B \setminus \{b_1, \ldots, b_k\}))\sigma \ ,$$

con σ l'unificatore più generale per L, è detta *risolvente*.

Risoluzione al primo ordine

Teorema 12.10

Un insieme finito di clausole al prim'ordine è insoddisfacibile se e solo se $S \vdash_{\mathsf{res}\bot}$.

La dimostrazione è omessa.

Riferimenti

Questa lezione è tratta principalmente dal capitolo 6 di *Andrea Asperti* e *Agata Ciabattoni*, Logica a Informatica, McGraw-Hill (1997).

Logica

Lezione 13

Dr Marco Benini

marco.benini@uninsubria.it

Dipartimento di Scienza e Alta Tecnologia
Università degli Studi dell'Insubria

a.a. 2020/21

Syllabus

Logica al prim'ordine:

- Programmazione logica (esempi)

Programmazione logica

La regola d'inferenza della risoluzione al prim'ordine può essere usata per costruire un sistema di programmazione, basato su un paradigma differente dall'usuale programmazione procedurale o orientata agli oggetti.

Questo paradigma alternativo, ha dato origine ad un linguaggio di programmazione effettivamente implementato: il *Prolog*.

Clausole

Una clausola al prim'ordine è una disgiunzione di letterali, e un letterale è una formula atomica, eventualmente negata. Raggruppando i letterali positivi e negativi di una clausola C, otteniamo che essa può essere scritta come

$$C = (\bigvee_i a_i) \vee (\bigvee_j \neg b_j)$$

con gli a e b formule atomiche.

Applicando le leggi di De Morgan, possiamo riscrivere C in modo equivalente come

$$C = (\bigvee_i a_i) \vee \neg(\bigwedge_j b_j) \ .$$

Quindi, interpretando la clausola come una implicazione,

$$C = (\bigwedge_j b_j) \supset (\bigvee_i a_i) \ .$$

Clausole

Se ci limitiamo a considerare le clausole C con al massimo un letterale positivo, otteniamo quattro casi

- le clausole formate da un solo letterale positivo, $C = a$;
- le clausole che contengono un letterale positivo e un certo numero $n > 0$ di letterali negativi, $C = b_1 \wedge \cdots \wedge b_n \supset a$;
- le clausole formate solamente da un numero $n > 0$ di letterali negativi, $C = \neg(b_1 \wedge \cdots \wedge b_n)$;
- la clausola priva di letterali, $C = \bot$.

Interpretazione dichiarativa

Queste clausole possono essere interpretate come segue:

- $C = a$ è una *asserzione*;
- $C = b_1 \wedge \cdots \wedge b_n \supset a$ è una *regola*;
- $C = \neg(b_1 \wedge \cdots \wedge b_n)$ è interpretata come la *domanda* $b_1 \wedge \cdots \wedge b_n$;
- $C = \bot$ è, semplicemente, la contraddizione.

Se immaginiamo che un programma sia formato da asserzioni e regole, e immaginiamo che una richiesta di computazione sia formata da una domanda, abbiamo un paradigma di programmazione puramente logico.

Interpretazione dichiarativa

Esempio 13.1

Consideriamo il *programma*:

$$\text{uomo}(\text{socrate})$$
$$\text{uomo}(x) \supset \text{mortale}(x)$$

dove x è una variabile, a fronte della *domanda* mortale(socrate), ci attendiamo che il programma calcoli la risposta *Si*.

Infatti, per soddisfare mortale(socrate), in base alla conoscenza del programma, è necessario che sia soddisfatto uomo(socrate), il che è esattamente il primo fatto.

Computazione

Consideriamo un programma come una lista di asserzioni e regole. Diremo che una asserzione A *unifica mediante* σ con un letterale B se σ è l'unificatore più generale che renda uguali A e B. Diremo che una regola $A_1 \wedge \cdots \wedge A_n \supset A$ *unifica mediante* σ con il letterale B se σ è l'unificatore più generale che renda uguali A e B.

Considereremo un programma come una lista di asserzioni e regole. E, a fronte di una domanda Q, l'esecuzione del programma sulla domanda può fallire, oppure avere successo, riportando come risultato una sostituzione.

Computazione

Sia dato un programma $P = p_1; \ldots; p_k$ e una domanda $Q = q_1 \wedge \cdots \wedge q_n$.

La computazione parte da q_1 con θ la sostituzione identica.

- se non esiste $p_i \in P$ che unifichi mediante un qualche σ con q_1, la computazione fallisce.
- se q_1 unifica con l'asserzione $p_i \in P$ mediante σ, si pone $\theta = \sigma \circ \theta$; se $n = 1$ allora la computazione riesce con output θ. Altrimenti, la computazione reinizia con la domanda $Q = q_2\sigma \wedge \cdots \wedge q_n\sigma$.
- se q_1 unifica con la regola $b_1 \wedge \cdots \wedge b_m \supset c \in P$ mediante σ, allora si pone $\theta = \sigma \circ \theta$, e si reinizia con $Q = b_1\sigma \wedge \cdots \wedge b_m\sigma \wedge q_2\sigma \wedge \cdots \wedge q_n\sigma$.

Stabiliamo il programma venga sempre scandito in ordine, e che a fronte di un fallimento, torniamo indietro di un passo (*backtracking*), dimenticando la σ, e provando, se esiste, la prima alternativa successiva. Se tale alternativa non esiste, la computazione fallisce.

Computazione

Esaminando in dettaglio la procedura precedente, vediamo che ogni passo di computazione corrisponde all'applicazione di una istanza della regola di risoluzione con unificazione.

Quindi, l'esito della computazione è

- un fallimento se il programma con la domanda è insoddisfacibile,
- un successo con esito σ se il programma con la domanda è soddisfatto dalla sostituzione σ.

Nulla impedisce che vi possano essere altre sostituzioni che soddisfano il programma con la domanda: esse possono essere ottenute continuando la computazione dopo un successo, mediante backtracking.

Esempi

Esempio 13.2

Programma

uomo(socrate)

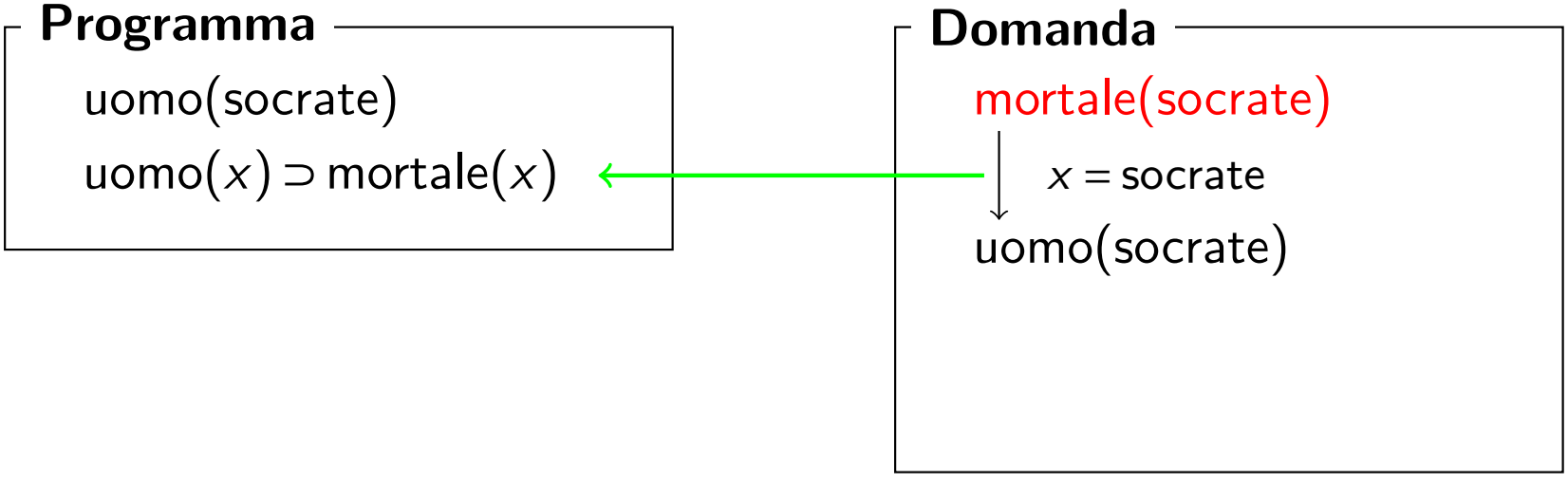
$\text{uomo}(x) \supset \text{mortale}(x)$

Domanda

mortale(socrate)

Esempi

Esempio 13.2



Esempi

Esempio 13.2

Programma

uomo(socrate)

uomo(x) $\supset$ mortale(x)

Domanda

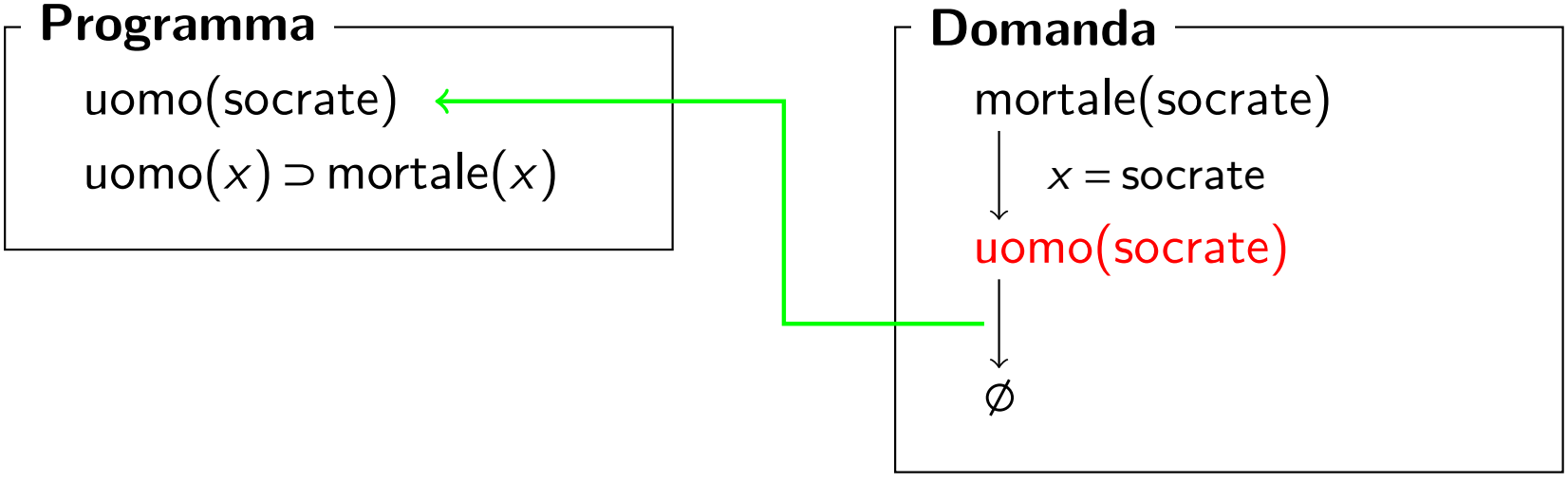
mortale(socrate)

$\downarrow$ x = socrate

uomo(socrate)

Esempi

Esempio 13.2



Esempi

Esempio 13.2

Programma

uomo(socrate)

uomo(x) $\supset$ mortale(x)

Domanda

mortale(socrate)

$\downarrow$ $x =$ socrate

uomo(socrate)

$\downarrow$

$\emptyset$ $\{x =$ socrate$\}$

Esempi

Esempio 13.3

Programma

uomo(socrate)

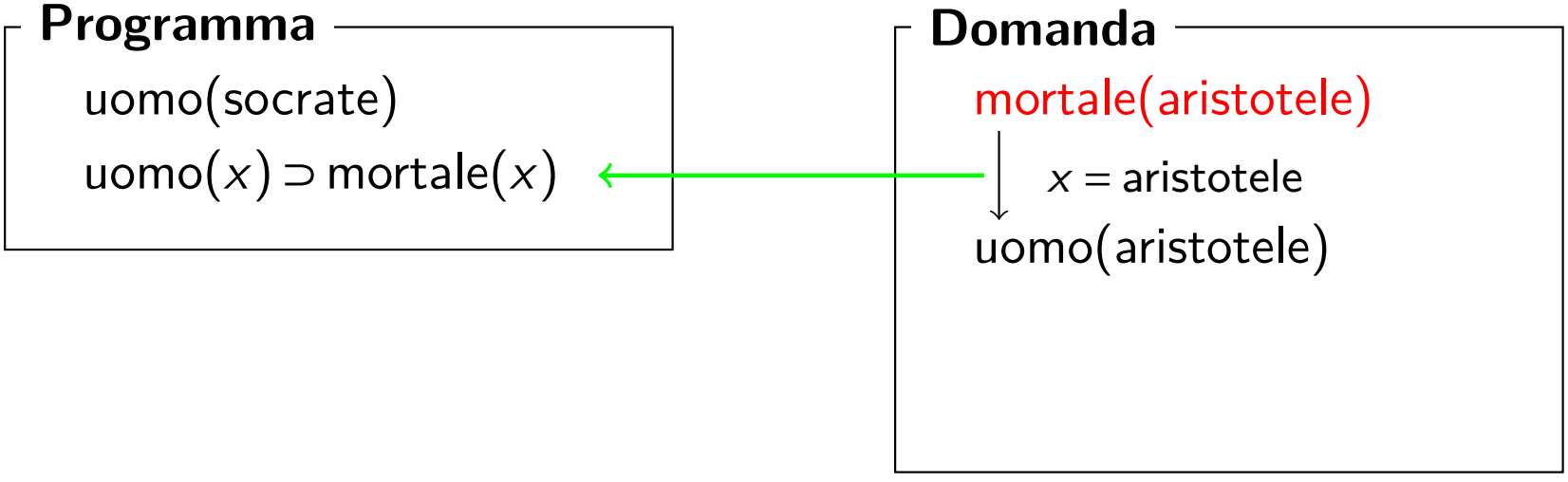
$\text{uomo}(x) \supset \text{mortale}(x)$

Domanda

mortale(aristotele)

Esempi

Esempio 13.3



Esempi

Esempio 13.3

Programma

uomo(socrate)

uomo(x) $\supset$ mortale(x)

Domanda

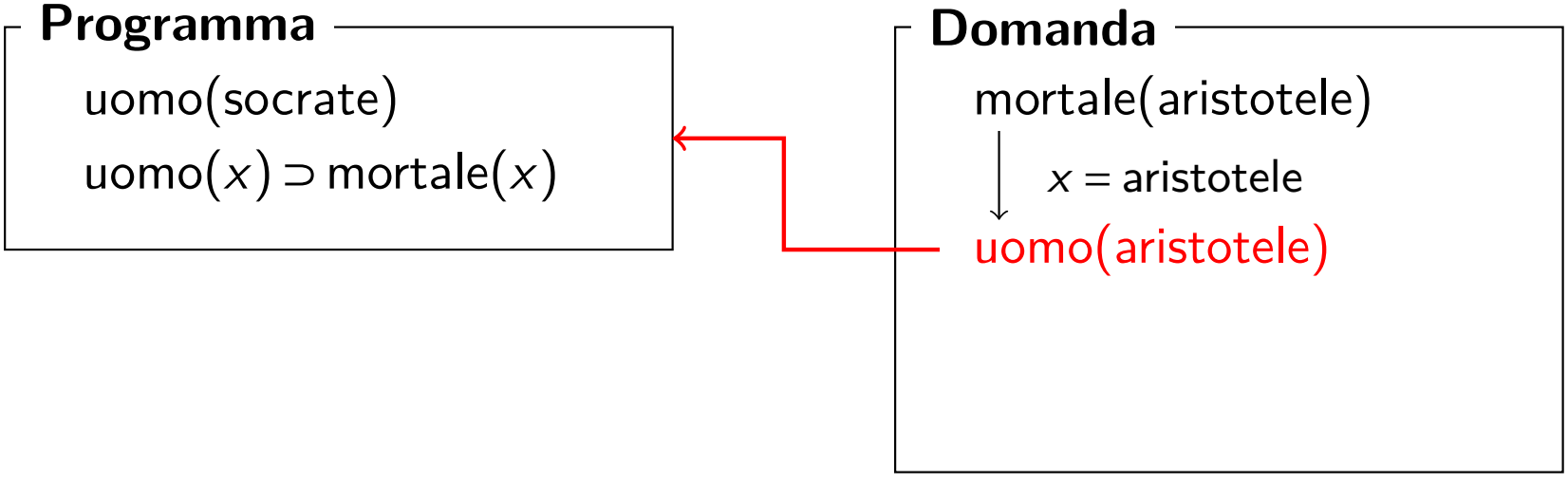
mortale(aristotele)

$\downarrow$ x = aristotele

uomo(aristotele)

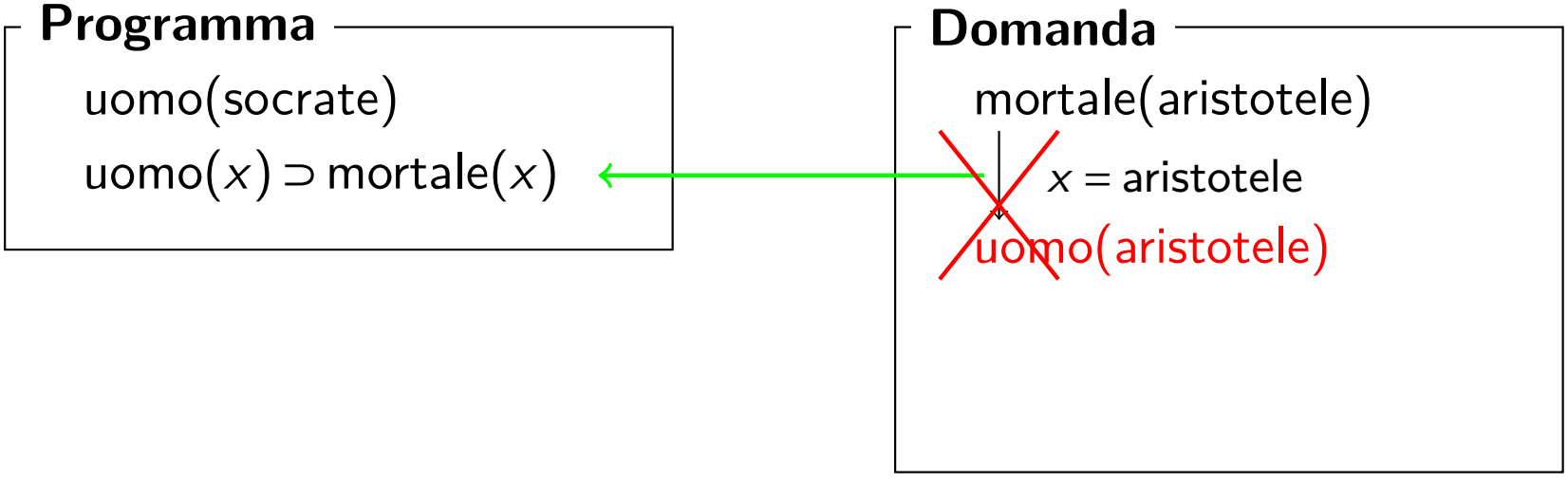
Esempi

Esempio 13.3



Esempi

Esempio 13.3



Esempi

Esempio 13.3

Programma

uomo(socrate)

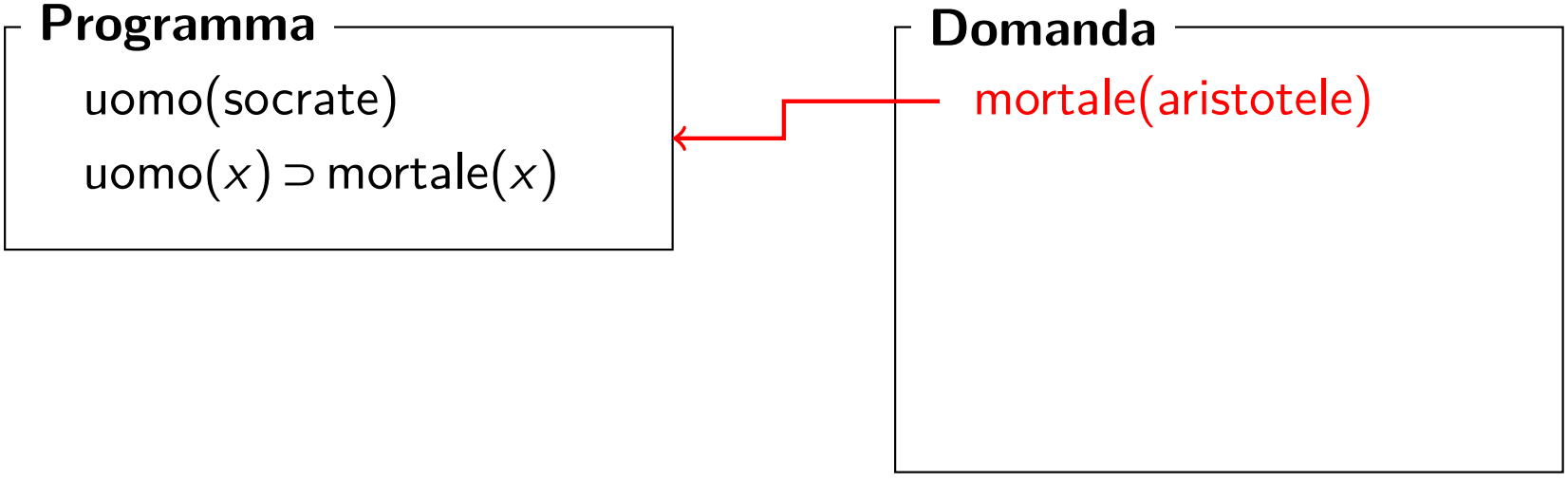
$\text{uomo}(x) \supset \text{mortale}(x)$

Domanda

mortale(aristotele)

Esempi

Esempio 13.3



Esempi

Esempio 13.3

Programma

uomo(socrate)

$\text{uomo}(x) \supset \text{mortale}(x)$

Domanda

mortale(aristotele)

Esempi

Esempio 13.4

Programma

$\text{padre}(x, y) \wedge \text{padre}(y, z) \supset \text{nonno}(x, z)$

$\text{padre}(\text{mario}, \text{carlo})$

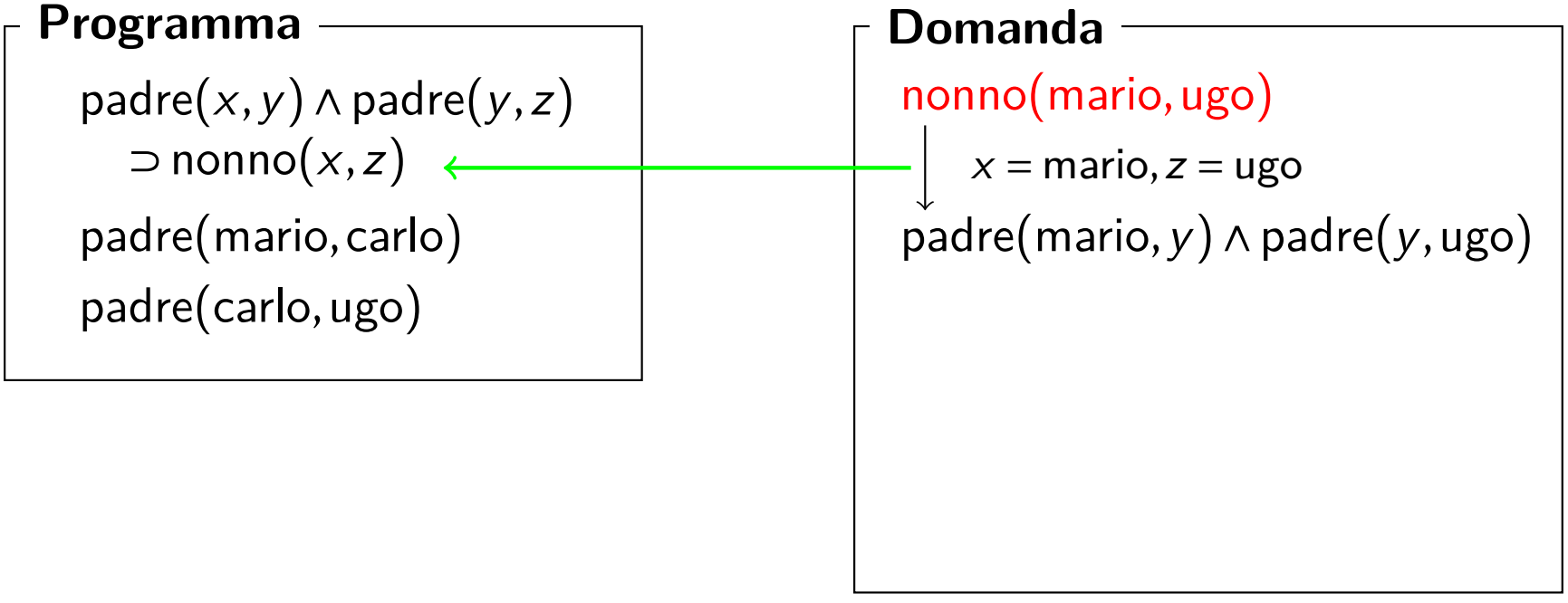
$\text{padre}(\text{carlo}, \text{ugo})$

Domanda

$\text{nonno}(\text{mario}, \text{ugo})$

Esempi

Esempio 13.4



Esempi

Esempio 13.4

Programma

$\text{padre}(x, y) \wedge \text{padre}(y, z) \supset \text{nonno}(x, z)$

$\text{padre}(\text{mario}, \text{carlo})$

$\text{padre}(\text{carlo}, \text{ugo})$

Domanda

$\text{nonno}(\text{mario}, \text{ugo})$

$\downarrow \quad x = \text{mario}, z = \text{ugo}$

$\text{padre}(\text{mario}, y) \wedge \text{padre}(y, \text{ugo})$

Esempi

Esempio 13.4

Programma

$\text{padre}(x, y) \wedge \text{padre}(y, z)$
$\supset \text{nonno}(x, z)$

$\text{padre}(\text{mario}, \text{carlo})$

$\text{padre}(\text{carlo}, \text{ugo})$

Domanda

$\text{nonno}(\text{mario}, \text{ugo})$

$\downarrow \quad x = \text{mario}, z = \text{ugo}$

$\text{padre}(\text{mario}, y) \wedge \text{padre}(y, \text{ugo})$

$\downarrow \quad y = \text{carlo}$

$\text{padre}(\text{carlo}, \text{ugo})$

Esempi

Esempio 13.4

Programma

$\text{padre}(x, y) \wedge \text{padre}(y, z) \supset \text{nonno}(x, z)$

$\text{padre}(\text{mario}, \text{carlo})$

$\text{padre}(\text{carlo}, \text{ugo})$

Domanda

$\text{nonno}(\text{mario}, \text{ugo})$

$\downarrow \quad x = \text{mario}, z = \text{ugo}$

$\text{padre}(\text{mario}, y) \wedge \text{padre}(y, \text{ugo})$

$\downarrow \quad y = \text{carlo}$

$\text{padre}(\text{carlo}, \text{ugo})$

Esempi

Esempio 13.4

Programma

$\text{padre}(x, y) \wedge \text{padre}(y, z) \supset \text{nonno}(x, z)$

$\text{padre}(\text{mario}, \text{carlo})$

$\text{padre}(\text{carlo}, \text{ugo})$

Domanda

$\text{nonno}(\text{mario}, \text{ugo})$

$\downarrow \quad x = \text{mario}, z = \text{ugo}$

$\text{padre}(\text{mario}, y) \wedge \text{padre}(y, \text{ugo})$

$\downarrow \quad y = \text{carlo}$

$\text{padre}(\text{carlo}, \text{ugo})$

$\downarrow$

$\emptyset$

Esempi

Esempio 13.4

Programma

$\text{padre}(x, y) \wedge \text{padre}(y, z) \supset \text{nonno}(x, z)$

$\text{padre}(\text{mario}, \text{carlo})$

$\text{padre}(\text{carlo}, \text{ugo})$

Domanda

$\text{nonno}(\text{mario}, \text{ugo})$

$\downarrow \quad x = \text{mario}, z = \text{ugo}$

$\text{padre}(\text{mario}, y) \wedge \text{padre}(y, \text{ugo})$

$\downarrow \quad y = \text{carlo}$

$\text{padre}(\text{carlo}, \text{ugo})$

$\downarrow$

$\emptyset \ \{x = \text{mario}, y = \text{carlo}, z = \text{ugo}\}$

Esempi

Esempio 13.5

Programma

$\text{padre}(x, y) \wedge \text{padre}(y, z) \supset \text{nonno}(x, z)$

$\text{padre}(\text{mario}, \text{carlo})$

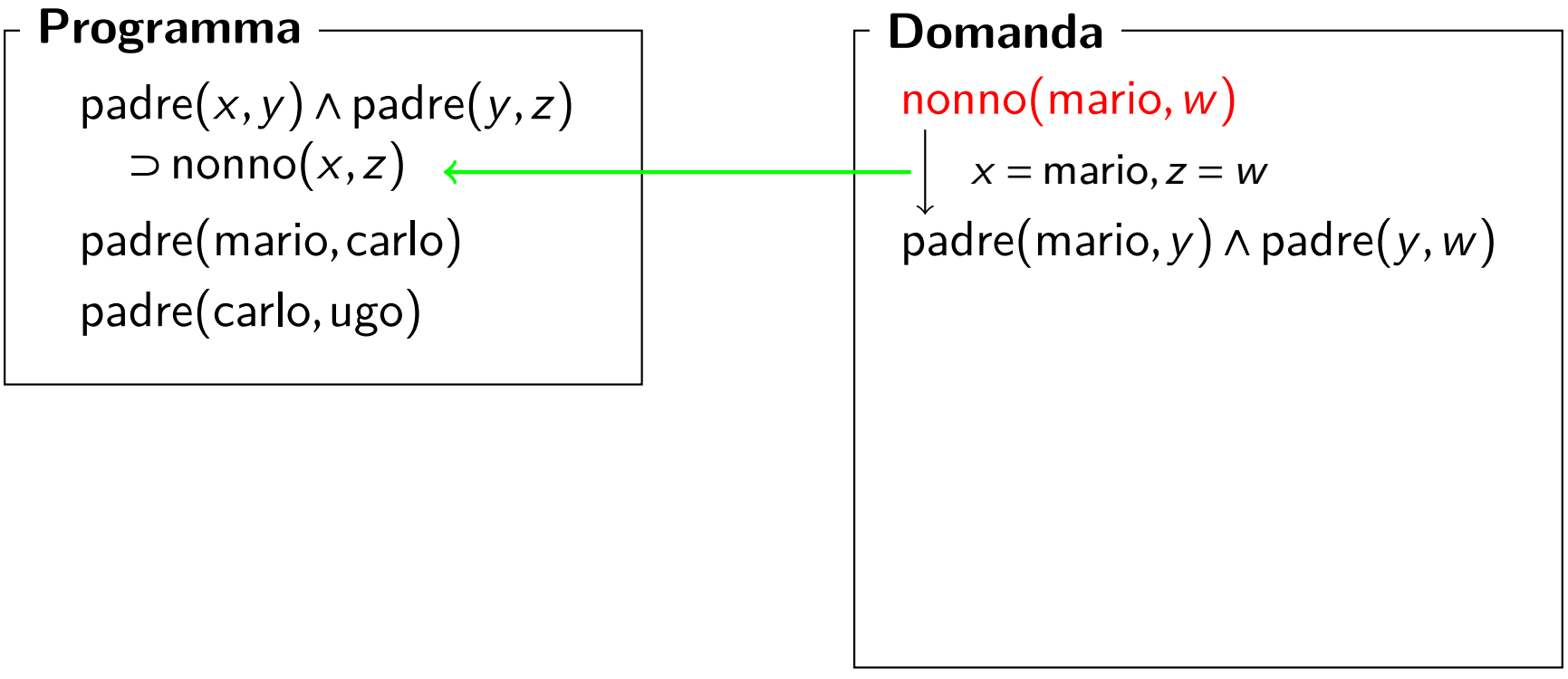
$\text{padre}(\text{carlo}, \text{ugo})$

Domanda

$\text{nonno}(\text{mario}, w)$

Esempi

Esempio 13.5



Esempi

Esempio 13.5

Programma

$\text{padre}(x, y) \wedge \text{padre}(y, z) \supset \text{nonno}(x, z)$

$\text{padre}(\text{mario}, \text{carlo})$

$\text{padre}(\text{carlo}, \text{ugo})$

Domanda

$\text{nonno}(\text{mario}, w)$

$\downarrow \quad x = \text{mario}, z = w$

$\text{padre}(\text{mario}, y) \wedge \text{padre}(y, w)$

Esempi

Esempio 13.5

Programma

$\text{padre}(x, y) \wedge \text{padre}(y, z)$
$\supset \text{nonno}(x, z)$

$\text{padre}(\text{mario}, \text{carlo})$

$\text{padre}(\text{carlo}, \text{ugo})$

Domanda

$\text{nonno}(\text{mario}, w)$

$\downarrow \quad x = \text{mario}, z = w$

$\text{padre}(\text{mario}, y) \wedge \text{padre}(y, w)$

$\downarrow \quad y = \text{carlo}$

$\text{padre}(\text{carlo}, w)$

Esempi

Esempio 13.5

Programma

$\text{padre}(x, y) \wedge \text{padre}(y, z) \supset \text{nonno}(x, z)$

$\text{padre}(\text{mario}, \text{carlo})$

$\text{padre}(\text{carlo}, \text{ugo})$

Domanda

$\text{nonno}(\text{mario}, w)$

$\downarrow \quad x = \text{mario}, z = w$

$\text{padre}(\text{mario}, y) \wedge \text{padre}(y, w)$

$\downarrow \quad y = \text{carlo}$

$\text{padre}(\text{carlo}, w)$

Esempi

Esempio 13.5

Programma

$\text{padre}(x, y) \wedge \text{padre}(y, z)$
$\quad \supset \text{nonno}(x, z)$

$\text{padre}(\text{mario}, \text{carlo})$

$\text{padre}(\text{carlo}, \text{ugo})$

Domanda

$\text{nonno}(\text{mario}, w)$

$\downarrow \quad x = \text{mario}, z = w$

$\text{padre}(\text{mario}, y) \wedge \text{padre}(y, w)$

$\downarrow \quad y = \text{carlo}$

$\text{padre}(\text{carlo}, w)$

$\downarrow \quad w = \text{ugo}$

$\emptyset$

Esempi

Esempio 13.5

Programma

$\text{padre}(x, y) \wedge \text{padre}(y, z) \supset \text{nonno}(x, z)$

$\text{padre}(\text{mario}, \text{carlo})$

$\text{padre}(\text{carlo}, \text{ugo})$

Domanda

$\text{nonno}(\text{mario}, w)$

$\downarrow \quad x = \text{mario}, z = w$

$\text{padre}(\text{mario}, y) \wedge \text{padre}(y, w)$

$\downarrow \quad y = \text{carlo}$

$\text{padre}(\text{carlo}, w)$

$\downarrow \quad w = \text{ugo}$

$\emptyset \; \{x = \text{mario} \;\; y = \text{carlo} \;\; z = w \;\; w = \text{ugo}\}$

Esempi

Esempio 13.6

Programma

$\text{padre}(x, y) \wedge \text{padre}(y, z) \supset \text{nonno}(x, z)$

$\text{padre}(\text{mario}, \text{carlo})$

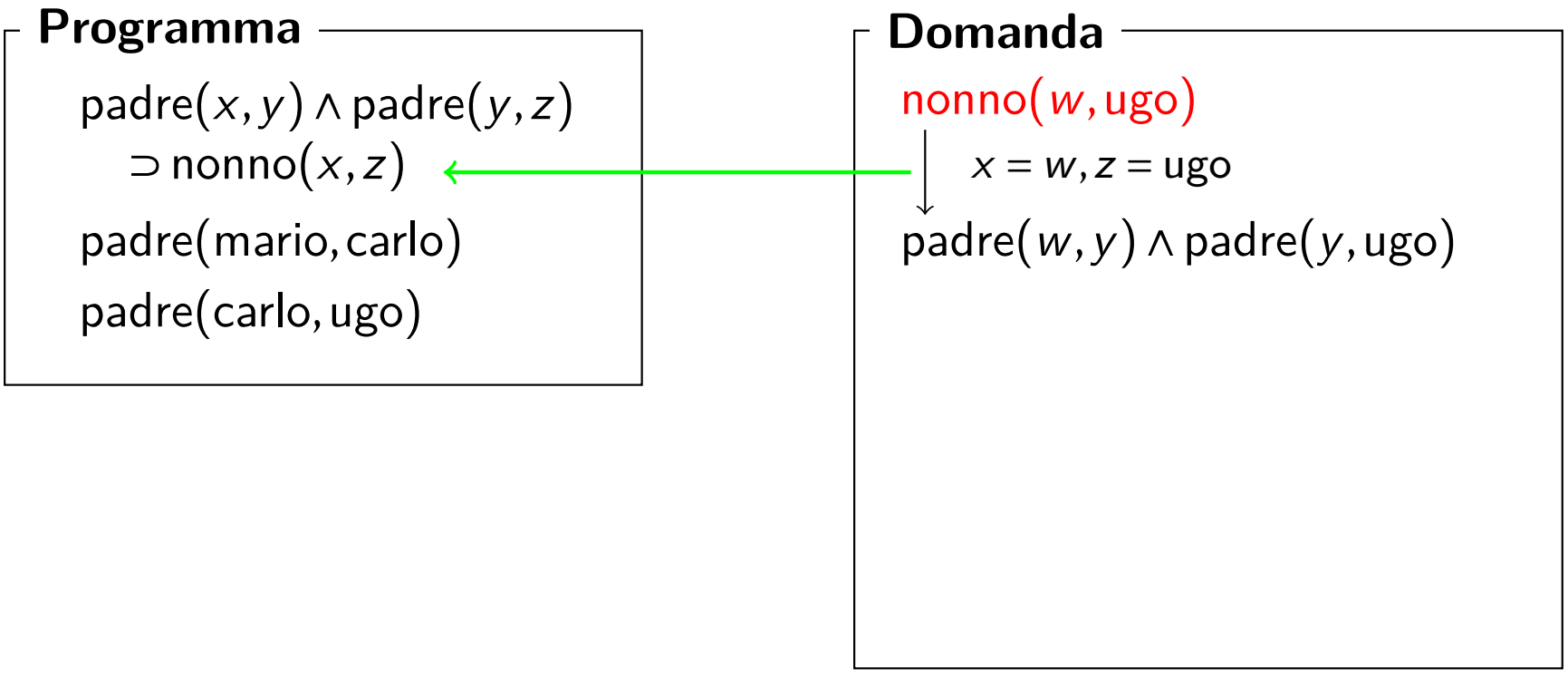
$\text{padre}(\text{carlo}, \text{ugo})$

Domanda

$\text{nonno}(w, \text{ugo})$

Esempi

Esempio 13.6



Esempi

Esempio 13.6

Programma

$\text{padre}(x, y) \wedge \text{padre}(y, z) \supset \text{nonno}(x, z)$

$\text{padre}(\text{mario}, \text{carlo})$

$\text{padre}(\text{carlo}, \text{ugo})$

Domanda

$\text{nonno}(w, \text{ugo})$

$\downarrow \quad x = w, z = \text{ugo}$

$\text{padre}(w, y) \wedge \text{padre}(y, \text{ugo})$

Esempi

Esempio 13.6

Programma

$\text{padre}(x, y) \wedge \text{padre}(y, z)$
$\quad \supset \text{nonno}(x, z)$

$\text{padre}(\text{mario}, \text{carlo})$

$\text{padre}(\text{carlo}, \text{ugo})$

Domanda

$\text{nonno}(w, \text{ugo})$

$\downarrow \quad x = w, z = \text{ugo}$

$\text{padre}(w, y) \wedge \text{padre}(y, \text{ugo})$

$\downarrow \quad y = \text{carlo}, w = \text{mario}$

$\text{padre}(\text{carlo}, \text{ugo})$

Esempi

Esempio 13.6

Programma

$\text{padre}(x, y) \wedge \text{padre}(y, z) \supset \text{nonno}(x, z)$

$\text{padre}(\text{mario}, \text{carlo})$

$\text{padre}(\text{carlo}, \text{ugo})$

Domanda

$\text{nonno}(w, \text{ugo})$

$\downarrow \quad x = w, z = \text{ugo}$

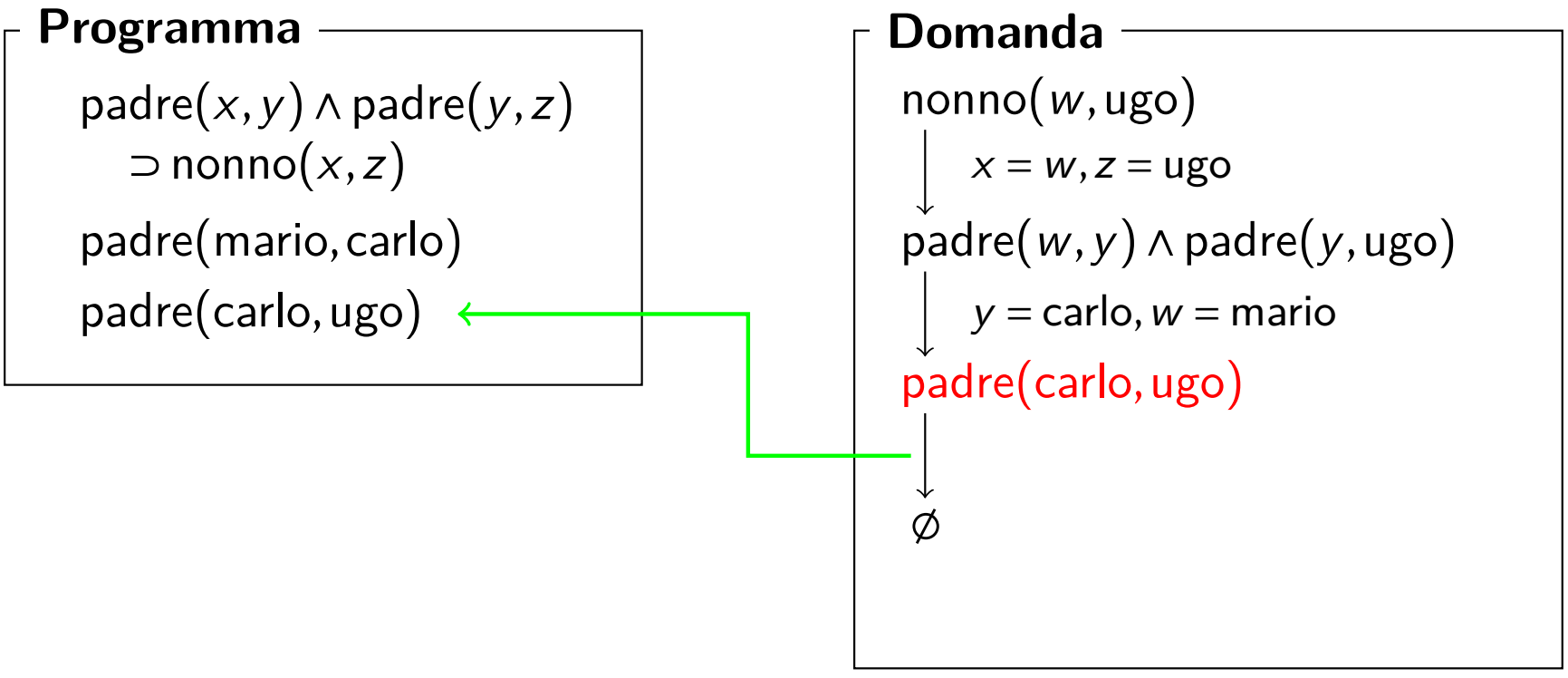
$\text{padre}(w, y) \wedge \text{padre}(y, \text{ugo})$

$\downarrow \quad y = \text{carlo}, w = \text{mario}$

$\text{padre}(\text{carlo}, \text{ugo})$

Esempi

Esempio 13.6



Esempi

Esempio 13.6

Programma

$\text{padre}(x, y) \wedge \text{padre}(y, z) \supset \text{nonno}(x, z)$

$\text{padre}(\text{mario}, \text{carlo})$

$\text{padre}(\text{carlo}, \text{ugo})$

Domanda

$\text{nonno}(w, \text{ugo})$

$\downarrow \quad x = w, z = \text{ugo}$

$\text{padre}(w, y) \wedge \text{padre}(y, \text{ugo})$

$\downarrow \quad y = \text{carlo}, w = \text{mario}$

$\text{padre}(\text{carlo}, \text{ugo})$

$\downarrow$

$\emptyset \; \{x = w \quad y = \text{carlo} \quad z = \text{ugo} \quad w = \text{mario}\}$

Riferimenti

Questa lezione è stata tratta da *Franco Furlan* e *Gaetano Aurelio Lanzarone*, Prolog: linguaggio e metodologia di programmazione logica, Franco Angeli (1988).

Logica

Lezione 14

Dr Marco Benini

marco.benini@uninsubria.it

Dipartimento di Scienza e Alta Tecnologia
Università degli Studi dell'Insubria

a.a. 2020/21

Syllabus

λ-calcolo puro

- Sintassi
- Riduzioni

λ-calcolo

Il λ-calcolo è una famiglia di sistemi formali, basati sul lavoro di Alonzo Church degli anni '30. Questi sistemi hanno lo scopo di descrivere le funzioni calcolabili usando una sintassi estremamente semplice. Sorprendentemente, non solo essi descrivono le funzioni calcolabili, ma, quando equipaggiati con un sistema di tipi, essi mostrano un legame nascosto e profondo tra la logica e la calcolabilità.

In questa lezione, vogliamo introdurre il λ-calcolo puro. Il nostro obiettivo è illustrare gli aspetti generali della teoria e derivare alcuni risultati fondamentali, che andremo ad usare nel resto del corso.

In molti casi, eviteremo di dimostrare i risultati che enunceremo. Questo è fatto di proposito: la semplicità del sistema formale ha come controparte naturale un profondo e complesso sviluppo tecnico. Sebbene questo sviluppo contenga numerose perle che gettano una luce profonda su importanti aspetti della calcolabilità, e diano una visione alternativa di cosa sia la programmazione, essi giacciono fuori dal contesto di questo corso.

λ-calcolo

Definizione 14.1 (λ-termine)

Fissato un insieme V di *variabili*, che sia al contempo infinito e ricorsivo, un λ-*termine* è induttivamente definito come:

- ogni $x \in V$ è un λ-termine, e $\mathrm{FV}(x) = \{x\}$;
- se M e N sono λ-termini, anche $(M \cdot N)$, detto *applicazione* è un λ-termine, e $\mathrm{FV}(MN) = \mathrm{FV}(M) \cup \mathrm{FV}(N)$;
- se $x \in V$ e M è un λ-termine, anche $(\lambda x.M)$, detto *astrazione*, è un λ-termine e $\mathrm{FV}(\lambda x.M) = \mathrm{FV}(M) \setminus \{x\}$.

L'insieme $\mathrm{FV}(M)$ è chiamato l'insieme delle *variabili libere* in M, e le variabili che occorrano in M ma non siano in $\mathrm{FV}(M)$ sono dette essere *legate*.

Esempio 14.2
$(\lambda x.x)$ è un λ-termine privo di variabili libere, che rappresenta la funzione identica.

λ-calcolo

Come al solito, per semplificare la notazione, introduciamo un certo numero di convenzioni:

- le parentesi più esterne non si scrivono: $\lambda x.x$ invece di $(\lambda x.x)$;
- una sequenza di astrazioni consecutive è raggruppata: $\lambda x, y.x \cdot y$ al posto di $\lambda x.(\lambda y.x \cdot y)$;
- consideriamo l'applicazione come un prodotto, omettendo il punto: xy invece di $x \cdot y$;
- assumiamo che l'applicazione associ a sinistra: xyz al posto di$(xy)z$.

Inoltre, usiamo il termine *combinatore* per indicare un λ-termine che non abbia variabili libere.

Esempio 14.3

I seguenti sono combinatori:

- $I \equiv \lambda x.x$;
- $K \equiv \lambda x, y.x$;
- $S \equiv \lambda x, y, z.(xz)(yz)$;
- $\Omega \equiv (\lambda x.xx)(\lambda x.xx)$.

Definizione 14.4 (Sostituzione)

Per ogni M, N λ-termini, e x variabile, $M[N/x]$ è la *sostituzione* di x con N in M, definita per induzione su M come:

- $x[N/x] \equiv N$;
- $y[N/x] \equiv y$, quando $x \not\equiv y$;
- $(PQ)[N/x] \equiv (P[N/x])(Q[N/x])$;
- $(\lambda x.P)[N/x] \equiv \lambda x.P$;
- $(\lambda y.P)[N/x] \equiv \lambda y.P[N/x]$, quando $x \not\equiv y$ e $y \notin \mathrm{FV}(N)$;
- $(\lambda y.P)[N/x] \equiv \lambda z.(P[z/y])[N/x]$, quando $x \not\equiv y$ e $y \in \mathrm{FV}(N)$ e $z \notin \mathrm{FV}(P) \cup \mathrm{FV}(N)$.

Nell'ultima clausola, la variabile z è detta essere *nuova*, ed è sempre possibile scegliere una z che soddisfa il requisito.

Riduzioni

Definizione 14.5 (α-equivalenza)

I λ-termini M e N sono *α-equivalenti*, $M =_\alpha N$, quando

- $M \equiv N$;
- $M \equiv PQ$, $N \equiv P'Q'$, e $P =_\alpha P'$ e $Q =_\alpha Q'$;
- $M \equiv \lambda x.P$, $N \equiv \lambda x.P'$, e $P =_\alpha P'$;
- $M \equiv \lambda x.P$ e $N \equiv \lambda y.P[y/x]$

Quindi, due λ-termini sono α-equivalenti quando essi differiscono soltanto per il nome delle variabili legate.

Riduzioni

È immediato verificare che l'α-equivalenza è una relazione di equivalenza, ma essa è anche una *congruenza* rispetto alla sostituzione:

Enunciato 14.6

Se $M =_\alpha M'$ *e* $N =_\alpha N'$, *allora* $M[N/x] =_\alpha M'[N'/x]$.

Riduzioni

Definizione 14.7 (β-riduzione)

La relazione binaria tra λ-termini $M \rhd_{1,\beta} N$, *M β-riduce a N in un passo*, vale se e solo se $M \equiv M'[(\lambda x.P) \cdot Q/z]$ e $N \equiv M'[(P[Q/x])/z]$.

Diciamo che *M β-riduce* a N, $M \rhd_{\beta} N$, quando esiste una sequenza finita $P_1, \ldots, P_n$ tale che $M \equiv P_1$, $N \equiv P_n$ e, per ogni $1 \leq i < n$, $P_i \rhd_{1,\beta} P_{i+1}$.

Nel λ-calcolo, la computazione è effettuata dalla β-riduzione.

Riduzioni

Definizione 14.8 (Forma β-normale)

Un termine N è detto essere in *forma β-normale* quando esso non contiene nessun sotto-termine della forma $(\lambda x.P)Q$.

Rispetto alla computazione, i, λ-termini in forma β-normale rappresentano i valori.

Confluenza

Teorema 14.9 (Church-Rosser)

Se $M \triangleright_\beta P$ e $M \triangleright_\beta Q$, allora esiste un λ-termine R tale che $P \triangleright_\beta R$ e $Q \triangleright_\beta R$.

Corollario 14.10

Se $M \triangleright_\beta N$ e N è una forma β-normale, allora N è unica a meno di α-equivalenze.

Il Teorema di Church-Rosser e il suo corollario affermano che, sebbene la computazione nel λ-calcolo sia non deterministica, il risultato, quando esiste, è univocamente determinato.

β-uguaglianza

Definizione 14.11 (β-uguaglianza)

Diciamo che P è β-equivalente a Q, $P =_\beta Q$, quando c'è una sequenza finita $R_1, \ldots, R_n$ tale che $P \equiv R_1$, $Q \equiv R_n$ e, per ogni $1 \leq i < n$, $R_i \rhd_{1,\beta} R_{i+1}$, oppure $R_{i+1} \rhd_{1,\beta} R_i$, oppure $R_i =_\alpha R_{i+1}$.

Punto fisso

Teorema 14.12 (Punto fisso)

Esiste un combinatore $\mathbf{Y}$ *tale che* $\mathbf{Y}x =_\beta x(\mathbf{Y}x)$.

Dimostrazione.

Sia $U \equiv \lambda u, x.x(uux)$, e sia $\mathbf{Y} \equiv UU$. Allora
$\mathbf{Y}x \equiv (\lambda u, x.x(uux))Ux \rhd_\beta (\lambda x.x(UUx))x \rhd_\beta x(UUx) \equiv x(\mathbf{Y}x)$. □

La dimostrazione del Teorema di Punto Fisso è dovuta ad Alan Turing.

Il Teorema di Punto Fisso afferma che ogni λ-termine, quando è pensato come una funzione, ha un punto fisso che può essere calcolato dal combinatore $\mathbf{Y}$.

Funzioni rappresentabili

Definizione 14.13 (Numerali)

Per ogni $n \in \mathbb{N}$, il *numerale di Church* $\overline{n}$ è un λ-termine induttivamente definito come:

- $\overline{0} = \lambda x, y.\, y$;
- $\overline{n+1} = \lambda x, y.\, x(\overline{n}xy)$.

Funzioni rappresentabili

Definizione 14.14 (Funzioni rappresentabili)

Sia $f : \mathbb{N}^k \to \mathbb{N}$ una funzione parziale. Un λ-termine F è detto *rappresentare* la funzione f quando

- per ogni $n_1, \ldots, n_k \in \mathbb{N}$, se $f(n_1, \ldots, n_k) = m$, allora $F\overline{n_1}, \ldots, \overline{n_k} = \overline{m}$;
- per ogni $n_1, \ldots, n_k \in \mathbb{N}$, se $f(n_1, \ldots, n_k)$ non è definita, allora $F\overline{n_1}, \ldots, \overline{n_k}$ non ha una forma β-normale.

Funzioni rappresentabili

Teorema 14.15

Ogni funzione calcolabile può essere rappresentata nel λ-calcolo.

La dimostrazione del Teorema è molto complessa, ben oltre gli obiettivi di questo corso. Pertanto la ometteremo, ma mostreremo alcuni esempi per giustificarne il senso.

Funzioni rappresentabili

Esempio 14.16

La funzione successore è rappresentata da $\lambda x, s, z.\, s\,(x\,s\,z)$.

La somma di naturali è rappresentata da $\lambda x, y, s, z.\, xs\,(y\,s\,z)$.

La moltiplicazione di naturali è rappresentata da $\lambda x, y, s.\, x\,(y\,s)$.

L'esponenziazione di naturali è rappresentata da $\lambda x, y.\, y\,x$

Funzioni rappresentabili

Esempio 14.17

I valori Booleani $\top$ e $\bot$ sono rappresentati come $\lambda x, y.y$ e $\lambda x, y.x$, rispettivamente.

Quindi 'if x then y else z' è rappresentata da $\lambda x, y, z.xzy$.

Infatti,
if $\overline{\bot}$ then A else $B \equiv (\lambda x, y, z.xzy)(\lambda x, y.x)AB =_\beta (\lambda y, z.(\lambda x, y.x)zy)AB =_\beta (\lambda y, z.z)AB =_\beta B$,
mentre
if $\overline{\top}$ then A else $B \equiv (\lambda x, y, z.xzy)(\lambda x, y.y)AB =_\beta (\lambda y, z.(\lambda x, y.y)zy)AB =_\beta (\lambda y, z.y)AB =_\beta A$.

Riferimenti

Una introduzione classica, ma ancora eccellente, al λ-calcolo è *J.R. Hindley* e *J.P. Seldin*, Lambda-Calculus and Combinators, Cambridge University Press, (2008).

Logica

Lezione 15

Dr Marco Benini

marco.benini@uninsubria.it

Dipartimento di Scienza e Alta Tecnologia
Università degli Studi dell'Insubria

a.a. 2020/21

Syllabus

λ-calcolo puro

- Tipi di dati in λ-calcolo
- Esempi

Rappresentazione

Per capire come le rappresentazioni descritte nella lezione precedente possano essere concepite, possiamo pensarle come computazioni su modelli logici.

Per esempio, i numeri naturali sono induttivamente definiti a partire dallo 0 e dal successore. Quindi un modello per i naturali è dato quando specifichiamo un insieme con un modo per interpretare il simbolo 0 in qualche elemento specifico, e il successore come una funzione iniettiva che trasforma un elemento in un altro.

Rappresentazione

Si consideri $\overline{0} \equiv \lambda x, y.\, y$: questa è una funzione dal modello che fornisce un elemento del modello. Il modello è rappresentato fornendo l'interpretazione del successore e dello zero. Il risultato è l'interpretazione di 0.

Si consideri $\overline{n+1} \equiv \lambda x, y.\, x(\overline{n}\, x\, y)$: poiché $\overline{n}$ trasforma un modello in un numero in quel modello, il termine $\overline{n}\, x\, y$ viene valutato in n nel modello dato da (x, y). Ma x è la funzione successore nel modello, quindi il risultato è il successore di n nel modello.

Rappresentazione

Consideriamo $x + y = \lambda x, y, s, z.\, x\, s\, (y\, s\, z)$: la somma è calcolata interpretando x nel modello specificato da s come successore e $(y\, s\, z)$ come zero, ovvero nello stesso modello ma con lo zero che corrisponde al valore di y.

In modo del tutto analogo, xy è calcolato interpretando x in un modello dove il successore si muove in avanti di y passi alla volta.

Strutture dati

Una struttura dati è un modo per rappresentare i dati in un programma. La maggior parte delle strutture dati possono essere codificate come una segnatura multi-sortata. Ogni istanza della struttura dati, quindi diviene un termine logico.

Definizione 15.1 (Struttura dati)

Una *struttura dati* D è una coppia $D = \langle S, F \rangle$ di insiemi finiti di simboli, dove S è detto l'insieme delle sorte e F l'insieme delle funzioni.
Ogni elemento $f \in F$ ha un *tipo*, $f : s_1 \times \cdots \times s_n \to s$ tale che $s, s_1, \ldots, s_n \in S$.

Una sorta $t \in S$ è *parametrica* se, per ogni $f : s_1 \times \cdots \times s_n \to s \in F$, $s \not\equiv t$.

Strutture dati

Ogni struttura dati D può essere rappresentata nel λ-calcolo. Ovvero ogni termine costruito a partire dal linguaggio D può essere scritto in modo unico come un λ-termine in un modo che ne consenta la manipolazione.

Definizione 15.2 (Rappresentazione)

Sia $D = \langle S, F \rangle$ una struttura dati, con $F = \{f_1, \ldots, f_n\}$. Un termine t su D ha la forma $f(t_1, \ldots, t_m)$ con $f : s_1 \times \cdots \times s_m \to s$ e $t_1, \ldots, t_m$ termini di tipo $s_1, \ldots, s_m$. Questo termine è rappresentato da un λ-termine $\underline{t} \equiv \underline{f}\,\underline{t_1} \ldots \underline{t_m}$ dove $f \equiv f_i$ per qualche indice i e $\underline{f} \equiv \lambda x_1, \ldots, x_m, f_1, \ldots, f_n.\, f_i\, \underline{s_1} \ldots \underline{s_m}$ con

$$\underline{s_i} \equiv \begin{cases} x_i & \text{se } s_i \text{ è una sorta parametrica} \\ (x_i\, f_1 \ldots f_n) & \text{altrimenti} \end{cases}$$

Booleani

I valori booleani corrispondono alla struttura dati

$$\mathsf{Bool} = \langle \{\mathsf{bool}\}, \{\mathsf{true}\colon \mathsf{bool}, \mathsf{false}\colon \mathsf{bool}\}\rangle$$

Definizione 15.3 (Booleani)

Codifichiamo i booleani seguendo la regola generale. Il risultato è

- $\mathsf{true} \equiv \lambda x, y.\, x$;
- $\mathsf{false} \equiv \lambda x, y.\, y$.

Qui e nel seguito omettiamo le sottolineature quando è chiaro dal contesto se stiamo parlando di termini logici o di λ-termini.

Booleani

Poiché nella nostra rappresentazione una costante $c : s \in F$ è rappresentata come $c \equiv \lambda f_1, \ldots, f_n . c$, ovvero come l'$i$-esimo proiettore, segue che

$$\mathsf{If} \equiv \lambda p, x, y . p\, x\, y$$

riduce ad una selettore quando il primo argomento è un valore booleano.

Infatti, $(\mathsf{If}\,\mathsf{true}\,a\,b) =_\beta a$ e $(\mathsf{If}\,\mathsf{false}\,a\,b) =_\beta b$.

Con un piccolo artificio sintattico, otteniamo la scrittura

$$\mathsf{if}\, c\, \mathsf{then}\, x\, \mathsf{else}\, y \equiv \mathsf{If}\, c\, x\, y \quad ,$$

più leggibile e presente nella maggior parte dei linguaggi funzionali. In essi il ramo 'else' è obbligatorio.

Booleani

Con questi strumenti, è facile definire le usuali operazioni sui booleani, ad esempio:

- $\mathsf{Conj} \equiv \lambda x, y.\, \mathsf{If}\, x\, y\, \mathsf{false}$;
- $\mathsf{Or} \equiv \lambda x, y.\, \mathsf{If}\, x\, \mathsf{true}\, y$;
- $\mathsf{Not} \equiv \lambda x.\, \mathsf{If}\, x\, \mathsf{false}\, \mathsf{true}$.

Enumerazioni

Le enumerazioni sono codificate dalla seguente struttura dati

$$\mathsf{Enum} = \langle \{\mathsf{enum}\}, \{\mathsf{e}_1 : \mathsf{enum}, \ldots, \mathsf{e}_n : \mathsf{enum}\} \rangle \ .$$

Definizione 15.4 (Enumerazioni)

Rappresentiamo gli elementi di una enumerazione come λ-termini seguendo la regola generale, ottenendo che $\mathsf{e}_i \equiv \lambda x_1, \ldots, x_n.x_i$.

Si noti come i booleani siano un caso speciale delle enumerazioni.

Enumerazioni

Come abbiamo fatto per i booleani, introduciamo un selettore per le enumerazioni:

$$\mathsf{Case} \equiv \lambda p, x_1, \ldots, x_n.\, p\, x_1 \ldots x_n \ ,$$

che ha la proprietà $\mathsf{Case}\, e_i\, a_1 \ldots a_n =_\beta a_i$.
Quindi la sintassi

case e
  e_1 : a_1
  ⋮
  e_n : a_n
end

è una presentazione leggibile del λ-termine $(\mathsf{Case}\, e\, a_1 \ldots a_n)$.

Prodotti cartesiani

I prodotti cartesiani sono codificati nella famiglia di strutture dati

$$A_1 \times \cdots \times A_n = \langle \{A_1, \ldots, A_n, T\}, \{\text{tuple}\colon A_1 \times \cdots \times A_n \to T\} \rangle \ .$$

Definizione 15.5 (Prodotto cartesiano)

Codifichiamo il costruttore tuple come un λ-termine seguendo la regola generale: $\text{tuple} \equiv \lambda x_1, \ldots, x_n, y.\, y\, x_1 \ldots x_n$.

Definendo $i\text{-th} \equiv \lambda y.\, y(\lambda x_1, \ldots, x_n.\, x_i)$, è immediato ottenere

$$i\text{-th}(\text{tuple}\, x_1 \ldots x_n) =_\beta x_i \ .$$

Poiché il tipo di dato dei record è pensabile come un prodotto cartesiano indicizzato da una enumerazione, la sua rappresentazione è immediata.

Numeri naturali

Anche la rappresentazione dei numeri naturali segue lo stesso schema

$$\mathsf{Nat} \equiv \langle \{N\}, \{\mathsf{Succ}\colon N \to N, 0\colon N\} \rangle \ .$$

Definizione 15.6 (Numeri naturali)

- $0 \equiv \lambda x, y.\, y$;
- $\mathsf{Succ} \equiv \lambda x, y, z.\, y\,(x\,y\,z)$.

Numeri naturali

Sia $f^n(x)$ l'abbreviazione di $f(\ldots(f\,x)\ldots)$ dove f occorre n volte. Allora, il numero k è scritto come il λ-termine $k \equiv \mathsf{Succ}^k(0) =_\beta \lambda x, y.\, x^k(y)$ come è facile provare per induzione su k.

Verificare se un numero è uguale a 0 è facile:

$$\mathsf{IsZero} \equiv \lambda n.\, n\,(\lambda x.\,\mathsf{false})\,\mathsf{true}\ ,$$

che soddisfa la proprietà $\mathsf{IsZero}\,0 =_\beta \mathsf{true}$ e $\mathsf{IsZero}(\mathsf{Succ}\,n) =_\beta \mathsf{false}$.

Numeri naturali

Le operazioni sui numeri naturali, come già visto, sono definite come

- $\mathsf{Add} \equiv \lambda x, y, u, v.\, x\, u\, (y\, u\, v)$;
- $\mathsf{Mult} \equiv \lambda x, y, z.\, x\, (y\, z)$;
- $\mathsf{Exp} \equiv \lambda x, y, u, v.\, y\, x\, u\, v$.

Liste

La struttura dati delle liste su un tipo A è

$$\mathrm{List}(A) \equiv \langle \{A, L\}, \{\mathrm{cons}\colon A \times L \to L, \mathrm{nil}\colon L\} \rangle \ .$$

Definizione 15.7 (Lista)

Seguendo lo schema generale

- $\mathrm{nil} \equiv \lambda x, y.\, y$;
- $\mathrm{cons} \equiv \lambda x, y, u, v.\, u\, x\, (y\, u\, v)$.

La forma generale di una lista è

$$[a_1, \ldots, a_n] \equiv \mathrm{cons}\, a_1\, (\mathrm{cons}\, a_2\, \ldots\, (\mathrm{cons}\, a_n\, \mathrm{nil}) \ldots) \ ,$$

che diviene, dopo una appropriata β-riduzione, $\lambda x, y.\, x\, a_1\, (x\, a_2\, (\ldots (x\, a_n\, y) \ldots))$ come si verifica facilmente per induzione su n.

Liste

È semplice definire l'operazione che estrae il primo elemento di una lista:

$$\mathsf{hd} \equiv \lambda x.\, x\,(\lambda u, v.\, u)\,\mathsf{nil}\ ,$$

e vale che $\mathsf{hd}(\mathsf{cons}\, a\, l) =_\beta a$.

D'altro canto, definire Null e tl tali che $\mathsf{Null}\,\mathsf{nil} =_\beta \mathsf{true}$ e $\mathsf{Null}\,(\mathsf{cons}\, a\, l) =_\beta \mathsf{false}$, e anche $\mathsf{tl}\,(\mathsf{cons}\, a\, l) =_\beta l$, non è immediato, e richiede sviluppare qualche risultato oltre quanto presentato in questo corso. Essenzialmente, serve poter scrivere un ciclo che scandisca tutti gli elementi della lista. Questo è possibile, usando il teorema di punto fisso, ma non facile.

In generale

Da questi esempi, risulta evidente che tutte le usuali strutture dati dell'Informatica possono essere codificate nel λ-calcolo, seguendo le regole presentate.

In effetti, è anche possibile definire una costruzione generica che, data la struttura dati, deriva automaticamente la rappresentazione della sue istanze, e che genera anche i combinatori per smontare le istanze e per testarle. Questo tipo di automatismi è disponibile in varia forma in tutti i linguaggi di programmazione funzionale.

Riferimenti

La rappresentazione delle strutture dati come λ-termini è derivata da *A. Berarducci* e *C. Böhm*, Automatic Synthesis of Typed Lambda-Programs on Term Algebras, Theoretical Computer Science 39 (1985) 135–154.

Gli esempi presentati si possono ritrovare nella maggior parte dei testi dedicati alla programmazione funzionale, ad esempio, in *B.C. Pierce*, Types and Programming Languages, The MIT Press, (2002)

Logica

Lezione 16

Dr Marco Benini

marco.benini@uninsubria.it

Dipartimento di Scienza e Alta Tecnologia
Università degli Studi dell'Insubria

a.a. 2020/21

Syllabus

λ-calcolo puro

- Programmazione funzionale
- Esempi

Programmazione funzionale

Nella lezione precedente abbiamo visto come le strutture dati possano essere codificate nel λ-calcolo. L'effetto è che il λ-calcolo assomiglia ad un linguaggio di programmazione, in cui molti costrutti che ritroviamo in altri linguaggi divengono disponibili.

Tuttavia, vi è una differenza fondamentale: non esiste un concetto di *stato*. Le variabili non contengono un valore che può essere mutato, non esiste una operazione di assegnamento. E, soprattutto, le funzioni sono prive di effetti collaterali: sono funzioni 'matematiche', che a fronte dello stesso input forniscono sempre lo stesso risultato.

Si tratta di un differente paradigma di programmazione, detto *funzionale*. Esso presenta vantaggi e difetti. Il principale vantaggio, da cui derivano tutti gli altri, è una base matematica solida.

Programmazione funzionale

La base matematica data dal λ-calcolo è immediatamente utile: quando siamo in grado di scrivere un insieme di equazioni (primitive) ricorsive, esse hanno una soluzione minima, costituita da una funzione che si può scrivere come un λ-termine.

Questo implica che possiamo *definire* una funzione mediante un insieme di equazioni ricorsive. E questa è la pratica comune. Poi, sarà il compilatore a calcolare la soluzione effettiva e ad impiegarla nella computazione.

Mostreremo come questa semplice idea porti a poter trattare strutture dati infinite e a manipolarle in modo efficiente.

Strutture di controllo

Nel paradigma di programmazione procedurale, vi sono tre principali strutture di controllo: la sequenza, la selezione, e l'iterazione.

Nel λ-calcolo, sebbene queste strutture sia simulabili, si veda ad esempio la selezione e il tipo dei booleani, esse risultano innaturali.

La principale struttura di controllo è la ricorsione, Ed essa viene concepita come un modo per definire funzioni, piuttosto che un modo per calcolare.

Con questo spirito, introduciamo, mediante esempi, le più semplici tecniche di programmazione funzionale. Il valore di queste tecniche in un corso di Logica Matematica è doppio: servono a far comprendere come la teoria si possa tradurre in codice eseguibile; e servono a mostrare come programmare sia una attività formale, che può essere vista come una parte di una certa matematica.

Currying

Nella programmazione funzionale, funzioni come $\mathsf{Add}(x, y) = x + y$ sono tipicamente scritte come $\mathsf{Add}\, x\, y = x + y$.

Come λ-termini esse sono differenti:

- nel primo caso, $\mathsf{Add} : \mathbb{N}^2 \to \mathbb{N}$, e $\mathsf{Add} \equiv \lambda w : \mathbb{N}^2.\, (\mathsf{fst}\, w) + (\mathsf{snd}\, w)$;
- nel secondo caso, $\mathsf{Add} : \mathbb{N} \to (\mathbb{N} \to \mathbb{N})$ e $\mathsf{Add} \equiv \lambda x : \mathbb{N}, y : \mathbb{N}.\, x + y$.

Una funzione a più argomenti $f : A_1 \times \cdots \times A_n \to B$ può sempre essere scritta come una funzione $g : A_1 \to (\ldots (A_n \to B) \ldots)$ che calcola gli stessi valori. Questa tecnica si chiama *currying*.

Currying

La tecnica di currying è semplice, ma è anche potente. Infatti, se $\mathsf{Add}\, x\, y = x + y$, possiamo definire $\mathsf{Succ} \equiv \mathsf{Add}\, 1$. In generale, possiamo applicare parzialmente una funzione, per ottenere come risultato una funzione.

Trattare le funzioni come combinatori permette di inserirle in strutture dati, passarle come argomenti ad altre funzioni, ritornarle come risultato di una computazione, e possiamo costruire nuove funzioni durante l'esecuzione.

Quindi, in un linguaggo di programmazione funzionale, le funzioni sono oggetti di prima classe.

Liste

La principale struttura dati nella programmazione funzionale è data dalle liste.

Le liste possono essere manipolate a partire da alcune funzioni base, definite dalle seguenti equazioni:

- $\mathsf{Null} \equiv \mathsf{if}\,(L = [])\,\mathsf{then}\,\mathsf{true}\,\mathsf{else}\,\mathsf{false}$;
- $\mathsf{hd}\,(\mathsf{cons}\,X\,L) = X$;
- $\mathsf{tl}\,(\mathsf{cons}\,X\,L) = L$.

Liste

Funzioni più complesse sono costruite a partire da questi mattoni fondamentali.

Per esempio, take $L\,n$ ritorna i primi n elementi della lista L. Essa è definita come la soluzione della seguente coppia di equazioni.

$$\begin{aligned} \mathsf{take}\,[]\,i &= [] \\ \mathsf{take}\,(x :: xs)\,i &= \mathsf{if}\,(i > 0)\,\mathsf{then}\,x :: (\mathsf{take}\,xs\,(i-1))\,\mathsf{else}\,[] \quad . \end{aligned}$$

D'ora in poi scriveremo [] per nil, la lista vuota, e $x :: xs$ per $\mathsf{cons}\,x\,xs$.

Liste

L'opposto di take è drop, che scarta i primi n elementi da una lista data. Essa è definita come

$$\begin{aligned} \mathsf{drop}\,[]\,i &= [] \\ \mathsf{drop}\,(x :: xs)\,i &= \mathsf{if}\,(i > 0)\,\mathsf{then}\,(\mathsf{drop}\,xs\,(i-1))\,\mathsf{else}\,(x :: xs)\ . \end{aligned}$$

Liste

Concatenare due liste è l'operazione effettuata da append, usualmente scritta con la sintassi infissa @:

$$\begin{aligned} [] \mathbin{@} L &= L \\ (x :: xs) \mathbin{@} L &= x :: (xs \mathbin{@} L) \quad . \end{aligned}$$

Si noti come facendo currying della funzione append si ottiene la funzione che aggiunge una lista fissata in testa alla lista argomento.

Liste

Invertire una lista è facile. Mostriamo un algoritmo funzionale efficiente:

$$\begin{aligned} \mathsf{rev} &= \mathsf{revAux}\,[] \\ \mathsf{revAux}\,L\,[] &= L \\ \mathsf{revAux}\,L\,(x :: xs) &= \mathsf{revAux}\,(x :: L)\,xs \quad . \end{aligned}$$

Si noti come l'operazione di currying sia stata usata per definire rev da revAux. È spesso conveniente usare argomenti addizionali in una funzione per accumulare risultati intermedi, e poi scartarli mediante currying.

Questo algoritmo è efficiente in quanto usa la *ricorsione in coda*: la chiamata ricorsiva di revAux viene effettuata come ultimo passo, quindi può essere implementata mediante un ciclo. I compilatori di linguaggi funzionali riconoscono automaticamente questo schema, e ne danno una implementazione efficiente.

Funzionali

I *funzionali* sono funzioni che prendono altre funzioni come argomenti.

L'esempio più semplice è map:

$$\mathsf{map}\, f\, [x_1, \ldots, x_n] = [f(x_1), \ldots, f(x_n)] \ .$$

Il funzionale map è definito come

$$\begin{aligned} \mathsf{map}\, f\, [] &= [] \\ \mathsf{map}\, f\, (x :: xs) &= (f\, x) :: (\mathsf{map}\, f\, xs) \ . \end{aligned}$$

Funzionali

Un altro importante esempio di funzionale è filter. Dato un predicato p, ovvero una funzione che ritorni un valore booleano, e una lista L, esso ritorna la sottolista di L i cui elementi rendano p vera:

$$\begin{aligned} \text{filter}\, p\, [] &= [] \\ \text{filter}\, p\, (x :: xs) &= \text{if}\, (p\, x)\, \text{then}\, (x :: (\text{filter}\, p\, xs))\, \text{else}\, (\text{filter}\, p\, xs) \quad . \end{aligned}$$

Funzionali

Altri funzionali interessanti sulle liste sono Exists e Forall. Entrambi prendono come argomenti un predicato p e una lista L. Il primo funzionale, Exists, è vero quando esiste un elemento in L che soddisfi p, mentre il secondo, Forall, è vero quando ogni elemento in L soddisfa p.

$$\begin{aligned} \mathsf{Exists}\, p\, [] &= \mathsf{false} \\ \mathsf{Exists}\, p\, (x :: xs) &= (p\, x)\ \mathsf{Or}\ (\mathsf{Exists}\, p\, xs) \end{aligned}$$

$$\begin{aligned} \mathsf{Forall}\, p\, [] &= \mathsf{true} \\ \mathsf{Forall}\, p\, (x :: xs) &= (p\, x)\ \mathsf{Conj}\ (\mathsf{Forall}\, p\, xs)\ . \end{aligned}$$

Sequenze

L'idea che supporta la rappresentazione di liste infinite, o *sequenze* come sono solitamente chiamate, è usare le funzioni per generarne i valori. Tecnicamente, le sequenze sono la struttura dati definita da

$$\langle \{A, S\}, \{\mathsf{cons}\colon A \times (\mathbb{1} \to S) \to S, \mathsf{nil}\colon S\} \rangle \ .$$

dove $\mathbb{1}$ è il *tipo unitario*, che contiene un solo elemento, denotato da $()$.

Per enfatizzare la parentela con le liste, usiamo gli stessi costruttori.

Sequenze

Questo tipo di dati non segue il formato dei precedenti. Infatti, esso non è una algebra libera di termini.

Ad ogni modo, esso è facilmente rappresentabile adattando la procedura generale

$$\begin{aligned} \mathsf{cons} &\equiv \lambda x, y, u, v.\, u\, x\, (\mathbf{K}\, (y\, u\, v)) \\ \mathsf{nil} \quad &\equiv \lambda u, v.\, v \quad . \end{aligned}$$

dove il combinatore $\mathbf{K} \equiv \lambda x, y.\, x$ si presume che non venga ridotto se non quando applicato a due argomenti.

Sequenze

Per capire come la rappresentazione di sequenze funzioni, si consideri la lista $[a, b]$ rappresentata come una sequenza:

$$[a, b] \equiv \mathsf{cons}\, a\, (\mathsf{cons}\, b\, \mathsf{nil}) \rhd \lambda u, v.\, u\, a\, (\mathbf{K}\, (u\, b\, (\mathbf{K}\, v))) \ .$$

Si confronti con la rappresentazione di $[a, b]$ come lista:

$$[a, b] \equiv \mathsf{cons}\, a\, (\mathsf{cons}\, b\, \mathsf{nil}) \rhd \lambda u, v.\, u\, a\, (u\, b\, v) \ .$$

Sequenze

Sapendo che i sistemi di equazioni (primitivi) ricorsivi hanno una soluzione minima, è immediato definire per le sequenze

$$\begin{aligned}
\mathsf{hd}\,(\mathsf{cons}\,x\,xs) &= x \\
\mathsf{tl}\,(\mathsf{cons}\,x\,xs) &= xs \\
\mathsf{Null}\,\mathsf{nil} &= \mathsf{true} \\
\mathsf{Null}\,(\mathsf{cons}\,x\,xs) &= \mathsf{false}
\end{aligned}$$

Sequenze

Si consideri la funzione

$$\mathsf{From}\, k = \mathsf{cons}\, k\, (\mathbf{K}\, (\mathsf{From}\, (k+1)))\ ,$$

essa valuterebbe a

$$\mathsf{From}\, 1 \rhd \mathsf{cons}\, 1\, (\mathbf{K}\, (\mathsf{From}\, 2)) \rhd \cdots \rhd [1, 2, 3, \ldots, \mathbf{K}\, (\mathsf{From}\, n)] \rhd \ldots$$

Se assumiano che la strategia di riduzione non espanda $\mathbf{K}\, (\mathsf{From}\, 2)$, a meno che esso non abbia due argomenti, la riduzione termina al primo passo.

In questo modo, possiamo definire un termine che codifichi l'intera sequenza dei numeri naturali: $\mathsf{Nat} = \mathsf{From}\, 0$.

Sequenze

Con la possibilità di definire strutture dati infinite, come la sequenza dei numeri naturali, diviene necessario definire funzioni per ispezionarle e manipolarle. Queste funzioni sono analoghe a quelle sulle liste.

Ad esempio

$$\begin{aligned} \mathsf{take}\,0\,s &= [] \\ \mathsf{take}\,n\,(\mathsf{cons}\,x\,s) &= x :: (\mathsf{take}\,(n-1)\,(s\,())) \end{aligned}$$

che estrae la lista dei primi n elementi nella sequenza s.

Sequenze

La maggior parte dei funzionali sulle liste possono essere immediatamente ridefiniti per operare sulle sequenze:

$$
\begin{aligned}
\mathsf{map}\, f\, \mathsf{nil} &= \mathsf{nil} \\
\mathsf{map}\, f\, (\mathsf{cons}\, x\, s) &= \mathsf{cons}\, (f x)\, (\mathbf{K}\, (\mathsf{map}\, f\, (s\, ())))
\end{aligned}
$$

$$
\begin{aligned}
\mathsf{filter}\, p\, \mathsf{nil} &= \mathsf{nil} \\
\mathsf{filter}\, p\, (\mathsf{cons}\, x\, s) &= \mathsf{if}\, (p\, x)\, \mathsf{then}\, (\mathsf{cons}\, x\, (\mathbf{K}\, (\mathsf{filter}\, p\, (x\, ())))) \\
&\quad\ \mathsf{else}\, (\mathsf{filter}\, p\, (s\, ()))
\end{aligned}
$$

Numeri primi

Un numero naturale strettamente maggiore di 1 è detto essere *primo* se può essere diviso esattamente solo da 1 e da se stesso.

Un algoritmo che consente di derivare la sequenza di tutti i numeri primi è quello di Eratostene. Esso inizia con la sequenza dei numeri naturali maggiori o uguali a 2
Ad ogni passo, la testa di s, la sequenza in ingresso, è un numero primo p. La nuova sequenza da considerare è la coda di s in cui tutti i multipli di p siano cancellati.

Numeri primi

Dapprima, scriviamo una funzione che elimini i multipli di un dato numero da una sequenza:

$$\mathsf{sift}\, p = \mathsf{filter}\,(\lambda n.\, n \bmod p \neq 0) \ .$$

Quindi iteriamo la procedura di cancellazione dei multipli della testa dalla coda della sequenza:

$$\mathsf{sieve}\,(\mathsf{cons}\, p\, s) = \mathsf{cons}\, p\,(\mathbf{K}\,(\mathsf{sieve}\,(\mathsf{sift}\, p\,(s\,()))))\ .$$

Numeri primi

In questo modo, la sequenza dei numeri primi è definita e calcolata da

$$\mathsf{Primes} = \mathsf{sieve}\,(\mathsf{From}\,2) \quad .$$

e può essere ispezionata con take.

Ad esempio, $\mathsf{take}\,1000\,\mathsf{Primes}$ restituisce la lista dei primi mille numeri primi.

Riferimenti

Il materiale di questa lezione è adattato da *L.C. Paulson*, ML for the working programmer, 2ª edizione, Cambridge University Press (1996).

Logica

Lezione 17

Dr Marco Benini

marco.benini@uninsubria.it

Dipartimento di Scienza e Alta Tecnologia
Università degli Studi dell'Insubria

a.a. 2020/21

Syllabus

Teoria dei tipi semplici

- Sintassi
- Riduzione

Teoria dei tipi semplici

La teoria dei tipi semplici è stata introdotta da Alonzo Church nel 1940. Essa rappresenta una specializzazione del λ-calcolo puro in cui ad alcuni λ-termini sono associati dei *tipi*.

L'idea è semplicemente quella di associare ad ogni funzione, rappresentata come un'astrazione, il dominio e il codominio della funzione, e richiedere che l'algebra dei tipi faccia in modo che una funzione possa essere applicata solo ad un argomento del tipo corretto.

Sebbene la teoria dei tipi semplici sia molto elementare, le sue proprietà, che vedremo in seguito, permettono di stabilire un legame profondo con la logica, che ha conseguenze importanti nel campo delle applicazioni informatiche.

Sintassi

Definizione 17.1 (Tipo)

Fissato un insieme numerabile V_T di *variabili di tipo*, un *tipo* è induttivamente definito come segue:

- $x \in V_T$ è un tipo;
- 0 e 1 sono tipi;
- se α e β sono tipi, anche $(\alpha \times \beta)$, $(\alpha + \beta)$, e $(\alpha \to \beta)$ sono tipi.

Come al solito, omettiamo le parentesi che non sono strettamente necessarie: $\times$ lega più fortemente di $+$, e $+$ lega più fortemente di $\to$, così

$$\alpha \times \beta + \gamma \to (\alpha + \gamma) \times (\beta + \gamma)$$

sta per

$$((\alpha \times \beta) + \gamma) \to ((\alpha + \gamma) \times (\beta + \gamma)) \ .$$

Sintassi

Un tipo è usato per vincolare l'entità di interesse nella teoria dei tipi, il *termine*.

Definizione 17.2 (Termine)

Fissata una famiglia $\{V_\alpha\}_\alpha$ di *variabili*, disgiunta dall'insieme delle variabili per tipo, indicizzata dalla collezione dei tipi, tale che, per ogni α, V_α è numerabile e tale che $V_\alpha \cap V_\beta = \emptyset$ ogniqualvolta $\alpha \neq \beta$, un *termine* $t : \alpha$ di tipo α, assieme all'insieme delle sue *variabili libere*, è induttivamente definito come:

- se $x \in V_\alpha$ per qualche tipo α, $x : \alpha$ è un termine, e $\mathrm{FV}(x : \alpha) = \{x : \alpha\}$;
- $* : 1$ è un termine e $\mathrm{FV}(* : 1) = \emptyset$;
- per ogni tipo α, $\square_\alpha : 0 \to \alpha$ è un termine e $\mathrm{FV}(\square_\alpha : 0 \to \alpha) = \emptyset$;
- se $A : \alpha$ e $B : \beta$ sono termini, $\langle A, B\rangle : \alpha \times \beta$ è un termine e $\mathrm{FV}(\langle A, B\rangle : \alpha \times \beta) = \mathrm{FV}(A : \alpha) \cup \mathrm{FV}(B : \beta)$;
- se $A : \alpha \times \beta$ è un termine, anche $\pi_1 A : \alpha$ e $\pi_2 A : \beta$ sono termini, e $\mathrm{FV}(\pi_1 A : \alpha) = \mathrm{FV}(\pi_2 A : \beta) = \mathrm{FV}(A : \alpha \times \beta)$;

↪

Sintassi

↪ (Termine)

- se $A\colon \alpha$ è un termine allora, per ogni tipo β, $i_1^\beta A\colon \alpha+\beta$ e $i_2^\beta A\colon \beta+\alpha$ sono termini e $\mathrm{FV}(i_1^\beta A\colon \alpha+\beta) = \mathrm{FV}(i_2^\beta A\colon \beta+\alpha) = \mathrm{FV}(A\colon \alpha)$;
- se $C\colon \alpha+\beta$, $A\colon \alpha\to\gamma$ e $B\colon \beta\to\gamma$ sono termini, anche $\delta(C,A,B)\colon \gamma$ è un termine, e $\mathrm{FV}(\delta(C,A,B)\colon \gamma) = \mathrm{FV}(C\colon \alpha+\beta)\cup\mathrm{FV}(A\colon \alpha\to\gamma)\cup\mathrm{FV}(B\colon \beta\to\gamma)$;
- se $A\colon \beta$ è un termine e $x\in V_\alpha$, allora $\lambda x\colon \alpha.A\colon \alpha\to\beta$ è un termine e $\mathrm{FV}(\lambda x\colon \alpha.A\colon \alpha\to\beta) = \mathrm{FV}(A\colon \beta)\setminus\{x\colon \alpha\}$;
- se $A\colon \alpha$ e $B\colon \alpha\to\beta$ sono termini, allora $B\cdot A\colon \beta$ è un termine e $\mathrm{FV}(B\cdot A\colon \beta) = \mathrm{FV}(A\colon \alpha)\cup\mathrm{FV}(B\colon \alpha\to\beta)$.

I termini rappresentano le entità computazionali primitive: in pratica, le *istruzioni* del linguaggio di programmazione.

Riduzioni

I termini possono essere *ridotti* secondo le seguenti regole, dove si assume che entrambi i lati delle uguaglianze siano correttamente tipate:

- $\pi_1\langle A, B\rangle = A$;
- $\pi_2\langle A, B\rangle = B$;
- $\langle \pi_1 A, \pi_2 A\rangle = A$;
- $(\lambda x\colon \alpha. A)\cdot B = A[B/x]$, l'atto di sostituire B per x;
- $\lambda x\colon \alpha.(A\cdot x) = A$, quando $x\colon \alpha \notin \mathrm{FV}(A\colon \alpha \to \beta)$;
- $\delta(i_1 C, A, B) = A\cdot C$;
- $\delta(i_2 C, A, B) = B\cdot C$.

È chiaro che queste regole siano calcolabili: è facile scrivere delle semplici procedure che le calcolano in un qualsiasi linguaggio di programmazione.

Teoria dei tipi semplici

Se restringiamo i tipi a quelli generati dalle variabili di tipo, $\to$ e $\times$, e i termini, corrispondentemente, alle variabili e quelli della forma $\lambda x\colon \alpha . A\colon \alpha \to \beta$, dette *astrazioni*, $A \cdot B\colon \beta$, dette *applicazioni*, $\langle A, B\rangle\colon \alpha \times \beta$, dette *coppie*, $\pi_1 A\colon \alpha$ e $\pi_2 A\colon \beta$, dette *proiezioni*, otteniamo un sottosistema di particolare interesse.

Infatti, se interpretiamo $\times$ come il prodotto cartesiano, e $\to$ come lo spazio delle funzioni, possiamo facilmente derivare una rappresentazione per i numeri naturali, con le operazioni di somma, moltiplicazione ed esponenziazione, i valori booleani, la costruzione 'if-then-else', e così via.
Infatti, queste rappresentazioni non sono nient'altro che quelle derivate per il λ-calcolo puro, senza i tipi.

Tipi dipendenti

Una estensione della teoria dei tipi semplici cui vale la pena accennare, è quella dei *tipi dipendenti*.

L'idea è che possiamo concepire una estensione dell'idea di funzione e di prodotto cartesiano. Usualmente, se $f : A \to B$ e $a : A$, allora $f\,a : B$; anche, se $a : A$ e $b : B$, allora $\langle a, b\rangle : A \times B$.

Possiamo immaginare che il codominio di una funzione sia determinato dal valore del dominio: se $f : \Pi_{x:A} B$ e $a : A$, allora $f\,a : B[a/x]$. Analogamente, possiamo immaginare che il secondo elemento della coppia appartiene ad un tipo che dipende dal primo: se $a : A$ e $b : B[a/x]$, allora $\langle a, b\rangle : \Sigma_{x:A} B$.

Tipi dipendenti

Dal punto di vista informatico, questi tipi sono naturali: nel caso del prodotto cartesiano, è un *record con varianti*, in cui alcuni campi sono determinati dal valore di un campo precedente.

Non approfondiremo oltre la teoria dei tipi dipendenti. Ci basti dire che, con le opportune riduzioni, questa teoria diventa un sistema molto potente, che è alla base di gran parte dei più moderni sistemi di programmazione funzionale.

Riferimenti

Una classica e ottima introduzione alla teoria dei tipi semplici è *J.R. Hindley* e *J.P. Seldin*, Lambda-Calculus and Combinators, Cambridge University Press, (2008).

Logica

Lezione 18

Dr Marco Benini

marco.benini@uninsubria.it

Dipartimento di Scienza e Alta Tecnologia
Università degli Studi dell'Insubria

a.a. 2020/21

Syllabus

Logica proposizionale intuizionista

- Motivazione
- Sintassi
- Potenza espressiva

Motivazione

Consideriamo il seguente

Enunciato 18.1

Ci sono a e b numeri irrazionali tali che a^b è razionale.

Dimostrazione.

Siano $a = b = \sqrt{2}$. Allora $a^b = \sqrt{2}^{\sqrt{2}}$ è razionale oppure irrazionale. Nel primo caso, abbiamo dimostrato il risultato, altrimenti prendiamo $a = \sqrt{2}^{\sqrt{2}}$ e $b = \sqrt{2}$. Allora $a^b = \left(\sqrt{2}^{\sqrt{2}}\right)^{\sqrt{2}} = \sqrt{2}^2 = 2$. □

La dimostrazione è corretta ma insoddisfacente: alla fine non conosciamo una coppia di irrazionali che soddisfi la proprietà. Abbiamo una scelta tra due coppie di candidati ma nessun modo per decidere quale coppia soddisfi la nostra richiesta.

Motivazione

Al contrario la seguente dimostrazione è differente:

Dimostrazione.
Sia $a = \sqrt{2}$ e $b = \log_2 9$. È ben noto che a è irrazionale, ma anche b lo è. Infatti, se $\log_2 9 = m/n$ per qualche $m, n \in \mathbb{N}$ allora, per le proprietà del logaritmo, $2^m = 9^n$, che è impossibile poiché la parte sinistra dell'uguaglianza è pari, mentre la destra è dispari. Ma $a^b = \sqrt{2}^{\log_2 9} = 2^{(\log_2 9)/2} = 2^{\log_2 3} = 3$. □

In questo caso, l'enunciato dice che esistono due irrazionali a e b tali che a^b è razionale, e la dimostrazione fornisce una evidenza di ciò esibendo una coppia come richiesto.

Motivazione

In generale, vorremmo che ogni volta che dobbiamo dimostrare un enunciato della forma $A \vee B$ o $\exists x.P$, noi siamo in grado di indicare quale disgiunto valga tra A e B, oppure un valore per x. Inoltre, vorremmo che queste informazioni siano presenti nella dimostrazione.

Più precisamente, vorremmo che la dimostrazione di un enunciato di queste forme consista in un algoritmo che indichi il disgiunto vero oppure *costruisca* un valore per x.

Questa attitudine è perfettamente ragionevole e desiderabile, ma ha un prezzo: non possiamo più accettare di usare assiomi che violino direttamente questi requisiti. In particolare, vi sono assiomi nei sistemi classici che evidentemente sono incompatibili.
Infatti, la Legge del Terzo Escluso dice che $A \vee \neg A$ per ogni formula A, ma non fornisce alcun modo per decidere quale di queste due alternative mutuamente esclusive sia valida. Quindi, la Legge del Terzo Escluso deve essere rifiutata se adottiamo una nozione di *dimostrazione* come sopra.

Motivazione

La Legge del Terzo Escluso è essenziale nella prima prova dell'Enunciato 18.1: essa evita di decidere se $\sqrt{2}^{\sqrt{2}}$ sia razionale o meno (in effetti, non lo è).

Ma rigettare la Legge del Terzo Escluso non è sufficiente. C'è un gran numero di principi che pone problemi.
Per esempio, l'Assioma della Scelta. Una delle sue conseguenze, il Teorema di Tarski-Banach, afferma che è possibile tagliare una sfera in un numero finito di pezzi, che possono essere riassemblati tramite movimenti rigidi in modo tale da formare **due** sfere identiche a quella iniziale. La dimostrazione 'costruisce' i pezzi usando l'Assioma della Scelta. Tuttavia, chiunque non sia un matematico chiamerebbe questo un miracolo, a meno che non si mostri un **modo** per tagliare la sfera originale e un **modo** per riassemblarne i pezzi. E ogni matematico noterebbe che la dimostrazione fallisce proprio nel mostrare metodi **effettivi** per attuare queste operazioni.

Motivazione

Infatti, quanto vogliamo avere è un sistema logico che permette di *calcolare* gli oggetti o le scelte che dobbiamo fare. In un certo senso, siamo interessati a sistemi dove le dimostrazioni sono una sorta di algoritmo per costruire i risultati impliciti nella loro enunciazione.

Questa attitudine verso la matematica è chiamata *costruttivismo* e ha prodotto una differente sorta di sistemi logici. In questi sistemi, principi come la Legge del Terzo Escluso sono banditi o accettati sulla base del fatto che essi permettano o impediscano la possibilità di 'costruire' gli oggetti che i loro enunciati implicano esistere.

Vi sono molti sistemi costruttivi, e molte variazioni sul tema. Differenti fondamenti filosofici sono stati proposti per supportare gli approcci costruttivi, e vi sono gradazioni di costruttività nei sistemi logici e formali che dichiarano di aderire a questo paradigma.

Un fatto indisputabile è che la matematica costruttiva ha avuto, ha, e continuerà ad avere un profondo impatto sullo studio della calcolabilità e le sue applicazioni.

Logica intuizionista

Tra i molti sistemi costruttivi, la *logica intuizionista* occupa un posto speciale. Storicamente, è stato il primo tentativo di catturare in un sistema formale l'idea originale dell'approccio costruttivo alla matematica.
Praticamente, è il più semplice, il più studiato e, in un certo senso, il meglio compreso tra i sistemi in questa linea di pensiero.

Nel seguito introdurremo la logica intuizionista al prim'ordine, mostrando alcune delle sue caratteristiche principali. A differenza dello studio dei sistemi classici, non dimostreremo la maggior parte dei risultati e salteremo molti aspetti importanti: il campo della matematica costruttiva è ampio, profondo e complesso, e il nostro obiettivo è mostrare come un sistema non classico possa essere di interesse nell'ambito informatico.

Sintassi

Sintatticamente, la logica intuizionista è molto simile alla logica classica. Nel caso proposizionale, le formule sono formate esattamente nello stesso modo. Nel caso del prim'ordine, i termini e le formule sono costruiti identicamente.

La differenza risiede nella costruzione delle prove: le prove valide in logica intuizionista sono le prove classiche in deduzione naturale in cui non appaia la Legge del Terzo Escluso. In altre parole, il calcolo proposizionale e il calcolo al prim'ordine son identici al corrispondente calcolo classico eccetto per la regola della Legge del Terzo Escluso, che viene eliminata.

Potenza espressiva

Evidentemente, per definizione, ogni dimostrazione $\pi : \Gamma \vdash_T A$ in logica intuizionista, quindi senza usare il Terzo Escluso, è anche una dimostrazione valida in logica classica.

Quindi, potremmo pensare che la logica intuizionista sia meno espressiva della logica classica: è possibile che vi siano degli enunciati che siano dimostrabili classicamente, ma che non lo siano intuizionisticamente perché essi richiedono l'uso della Legge del Terzo Escluso in un modo essenziale. Al contrario, ogni risultato dimostrabile in logica intuizionista è anche valido in logica classica perché ogni dimostrazione intuizionista è anche una dimostrazione classica.

In un certo senso, questa osservazione è corretta.
Ma, in un altro senso, non lo è...

Potenza espressiva

. . . poiché l'abilità di dimostrare di più, avendo una regola d'inferenza addizionale, può portare a dimostrare l'inconsistenza di più teorie.

Per esempio, la tesi di Church nella teoria della calcolabilità afferma che una funzione $\mathbb{N} \to \mathbb{N}$ è calcolabile se e solo se esiste una macchina di Turing che la calcoli. Se noi diciamo che ogni funzione che possiamo definire nell'aritmetica è calcolabile, otteniamo quella che è chiamata la tesi formale di Church. Accade che la teoria formale dell'aritmetica con la tesi formale di Church sia una teoria perfettamente ragionevole dal punto di vista intuizionista, che può essere provata essere consistente rispetto all'aritmetica classica. Al contrario, la stessa teoria in logica classica risulta inconsistente.

Il motivo è semplice: in logica classica possiamo provare che esiste una funzione incomputabile, mostrando che assumere che ogni funzione sia calcolabile conduce ad una contraddizione. Quindi la tesi formale di Church, che afferma che ogni funzione è calcolabile, è contraddittoria con la teoria dell'aritmetica classica. In logica intuizionista, semplicemente, la prova che esista una funzione non computabile non può essere fatta, in quanto richiede il Terzo Escluso in modo essenziale.

Potenza espressiva

Da un altro punto di vista, in un certo senso, ogni teorema classico può essere provato in logica intuizionista, modulo una traduzione. Precisamente:

Definizione 18.2

La *traduzione di Gödel-Gentzen* è una mappa da formule in formule induttivamente definita come:

- $(\top)^N = \top$, $(\bot)^N = \bot$;
- per ogni A atomica, $(A)^N = \neg\neg A$;
- $(A \wedge B)^N = (A)^N \wedge (B)^N$;
- $(A \vee B)^N = \neg\left(\neg (A)^N \wedge \neg (B)^N\right)$;
- $(A \supset B)^N = (A)^N \supset (B)^N$;
- $(\forall x : s. A)^N = \forall x : s. (A)^N$;
- $(\exists x : s. A)^N = \neg\forall x : s. \neg (A)^N$.

Potenza espressiva

Enunciato 18.3

In logica classica, per ogni formula A, esiste una prova $\pi\colon \vdash A = (A)^N$.

Dimostrazione. (i)

Per induzione sulla formula A:

- $A \equiv \bot, \top$: $\vdash \bot \supset \bot$ e $\vdash \top \supset \top$ per introduzione di implicazione, quindi $\vdash \bot = \bot$ e $\vdash \top = \top$.
- A è atomica:

$$\dfrac{\dfrac{\dfrac{[A]^1 \quad [\neg A]^2}{\bot}\,\neg\mathrm{E}}{\neg\neg A}\,\neg\mathrm{I}^2}{A \supset \neg\neg A}\,\supset\mathrm{I}^1 \qquad \dfrac{\dfrac{}{A \vee \neg A}\,\mathrm{lem} \quad \dfrac{\dfrac{\dfrac{[\neg\neg A]^2 \quad [\neg A]^1}{\bot}\,\neg\mathrm{E}}{A}\,\bot\mathrm{E}}{A}\,\vee\mathrm{E}^1}{\neg\neg A \supset A}\,\supset\mathrm{I}^2$$

↪

Potenza espressiva

↪ Dimostrazione. (ii)

- $A \equiv B \wedge C$: per ipotesi induttiva esistono $\vdash B = (B)^N$ e $\vdash C = (C)^N$, e $(A)^N = (B)^N \wedge (C)^N$, quindi

$$\dfrac{\dfrac{\dfrac{[B \wedge C]^1}{B} \wedge \mathsf{E}_1 \quad \dfrac{[B \wedge C]^1}{C} \wedge \mathsf{E}_2}{\dfrac{(B)^N \qquad (C)^N}{(B)^N \wedge (C)^N} \wedge \mathsf{I}}}{B \wedge C \supset (B)^N \wedge (C)^N} \supset \mathsf{I}^1$$

$$\dfrac{\dfrac{\dfrac{[(B)^N \wedge (C)^N]^1}{(B)^N} \wedge \mathsf{E}_1 \quad \dfrac{[(B)^N \wedge (C)^N]^1}{(C)^N} \wedge \mathsf{E}_2}{\dfrac{B \qquad C}{B \wedge C} \wedge \mathsf{I}}}{(B)^N \wedge (C)^N \supset B \wedge C} \supset \mathsf{I}^1$$

↪

Potenza espressiva

↪ Dimostrazione. (iii)

- $A \equiv B \vee C$: per ipotesi induttiva esistono $\vdash B = (B)^N$ e $\vdash C = (C)^N$, e $(A)^N = \neg\left(\neg(B)^N \wedge \neg(C)^N\right)$, quindi

$$
\dfrac{\dfrac{[B \vee C]^1 \qquad \dfrac{[B]^2 \qquad \begin{array}{c}\dfrac{[(\neg(B)^N \wedge \neg(C)^N)]^3}{\neg(B)^N}\,{\scriptstyle \wedge E_1}\\ \vdots \\ \neg B\end{array}}{\bot}\,{\scriptstyle \neg E} \qquad \dfrac{[C]^2 \qquad \begin{array}{c}\dfrac{[(\neg(B)^N \wedge \neg(C)^N)]^3}{\neg(C)^N}\,{\scriptstyle \wedge E_2}\\ \vdots \\ \neg C\end{array}}{\bot}\,{\scriptstyle \neg E}}{\dfrac{\bot}{\neg\left(\neg(B)^N \wedge \neg(C)^N\right)}\,{\scriptstyle \neg I^3}}\,{\scriptstyle \vee E^2}}{B \vee C \supset \neg\left(\neg(B)^N \wedge \neg(C)^N\right)}\,{\scriptstyle \supset I^1}
$$

per le prove implicite, si veda il caso $\neg$.

↪

Potenza espressiva

↪ Dimostrazione. (iv)

$$\dfrac{\dfrac{}{B \vee \neg B}\,\text{lem} \quad \dfrac{[B]^1}{B \vee C}\,\vee\text{I}_1 \quad \dfrac{\dfrac{}{C \vee \neg C}\,\text{lem} \quad \dfrac{[C]^2}{B \vee C}\,\vee\text{I}_2 \quad \dfrac{\bot}{B \vee C}\,\bot\text{E}}{B \vee C}\,\vee\text{E}^2}{B \vee C}\,\vee\text{E}^1$$

$$\dfrac{B \vee C}{\neg\left(\neg(B)^N \wedge \neg(C)^N\right) \supset B \vee C}\,\supset\text{I}^3$$

$$\dfrac{\dfrac{\begin{array}{cc}[\neg B]^1 & [\neg C]^2 \\ \vdots & \vdots \\ \neg(B)^N & \neg(C)^N\end{array}}{\neg(B)^N \wedge \neg(C)^N}\,\wedge\text{I} \quad [\neg\left(\neg(B)^N \wedge \neg(C)^N\right)]^3}{}\,\neg\text{E}$$

per le prove implicite, si veda il caso $\neg$. ↪

Potenza espressiva

↪ Dimostrazione. (v)

- $A \equiv B \supset C$: per ipotesi induttiva esistono $\vdash B = (B)^N$ e $\vdash C = (C)^N$, e $(A)^N = (B)^N \supset (C)^N$, quindi

$$\dfrac{\dfrac{\dfrac{\dfrac{[(B)^N]^2}{\vdots}}{\dfrac{[B \supset C]^1 \quad B}{C}\supset E}}{\vdots}}{\dfrac{(C)^N}{\big((B)^N \supset (C)^N\big)}\supset I^2}}{(B \supset C) \supset \big((B)^N \supset (C)^N\big)}\supset I^1$$

$$\dfrac{\dfrac{\dfrac{\dfrac{[B]^2}{\vdots}}{\dfrac{[((B)^N \supset (C)^N)]^1 \quad (B)^N}{(C)^N}\supset E}}{\vdots}}{\dfrac{C}{B \supset C}\supset I^2}}{\big((B)^N \supset (C)^N\big) \supset (B \supset C)}\supset I^1$$

↪

Potenza espressiva

↪ Dimostrazione. (vi)

- $A \equiv \neg B$: per ipotesi induttiva esiste $\vdash B = (B)^N$, quindi

$$\dfrac{\dfrac{\dfrac{[(B)^N]^2}{\vdots}}{\dfrac{[\neg B]^1 \quad B}{\dfrac{\bot}{\dfrac{\neg (B)^N}{\neg B \supset \neg (B)^N}\supset \mathrm{I}^1}}\neg \mathrm{I}^2}}{}\qquad \dfrac{\dfrac{\dfrac{[B]^2}{\vdots}}{\dfrac{[\neg (B)^N]^1 \quad (B)^N}{\dfrac{\bot}{\dfrac{\neg B}{\neg (B)^N \supset \neg B}\supset \mathrm{I}^1}}\neg \mathrm{I}^2}}{}$$

↪

Potenza espressiva

↪ Dimostrazione. (vii)

- $A \equiv \forall x.B$: per ipotesi induttiva esiste $\vdash B = (B)^N$, e $(A)^N \equiv \forall x.(B)^N$ quindi

$$\dfrac{\dfrac{\dfrac{[\forall x.B]^1}{B}\,\forall \mathsf{E}}{\vdots}}{\dfrac{(B)^N}{\forall x.(B)^N}\,\forall \mathsf{I}}\,\supset \mathsf{I}^1 \qquad \dfrac{\dfrac{\dfrac{[\forall x.(B)^N]^1}{(B)^N}\,\forall \mathsf{E}}{\vdots}}{\dfrac{B}{\forall x.B}\,\forall \mathsf{I}}\,\supset \mathsf{I}^1$$

$$(\forall x.B) \supset \left(\forall x.(B)^N\right) \qquad \left(\forall x.(B)^N\right) \supset (\forall x.B)$$

↪

Potenza espressiva

↪ Dimostrazione. (viii)

- $A \equiv \exists x.B$: per ipotesi induttiva esiste $\vdash B = (B)^N$, e $(A)^N \equiv \neg\forall x.\neg(B)^N$ quindi

$$\dfrac{\dfrac{\dfrac{[\exists x.B]^1 \qquad \dfrac{\dfrac{[B]^2}{\vdots}}{(B)^N} \qquad \dfrac{[\forall x.\neg(B)^N]^3}{\neg(B)^N}\,\forall E}{\bot}\,\neg E}{\bot}\,\exists E^2}{\neg\forall x.\neg(B)^N}\,\neg I^3}{(\exists x.B) \supset \neg\forall x.\neg(B)^N}\,\supset I^1$$

↪

Potenza espressiva

↪ Dimostrazione. (ix)

$$\dfrac{\dfrac{\dfrac{}{(\exists x.\,B)\vee\neg(\exists x.\,B)}\,\text{lem} \qquad [\exists x.\,B]^1 \qquad \dfrac{\dfrac{\dfrac{\dfrac{\dfrac{[\neg(\exists x.\,B)]^1 \qquad \dfrac{\begin{matrix}[(B)^N]^2\\ \vdots\\ B\end{matrix}}{\exists x.\,B}\,\exists\text{I}}{\bot}\,\neg\text{E}}{\neg(B)^N}\,\neg\text{I}^2}{\forall x.\,\neg(B)^N}\,\forall\text{I} \qquad [\neg(\forall x.\,\neg(B)^N)]^3}{\bot}\,\neg\text{E}}{\exists x.\,B}\,\bot\text{E}}{\exists x.\,B}\,\vee\text{E}^1}{\neg\left(\forall x.\,\neg(B)^N\right)\supset\exists x.\,B}\,\supset\text{I}^3$$

□

Potenza espressiva

Enunciato 18.4

Se $\pi : \Gamma \vdash A$ *in logica classica, allora esiste* $\pi' : \left\{(\gamma)^N : \gamma \in \Gamma\right\} \vdash (A)^N$ *in logica intuizionista.*

Non dimostriamo questo risultato: chi fosse interessato può guardare i riferimenti alla fine di questa lezione.

L'enunciato ha numerose conseguenze: quelle rilevanti ai nostri fini sono

- ogni teoria classica e, quindi, ogni dimostrazione classica può essere tradotta in logica intuizionista, ottenendo un risultato classicamente equivalente. Quindi la logica classica non è davvero più espressiva della logica intuizionista.
- la logica intuizionista è più espressiva della logica classica in quanto consente di distinguere formule che sono classicamente equivalenti.

Riferimenti

Una buona introduzione al modo di ragionare costruttivo si può trovare in *A.S. Troelstra* e *D. van Dalen*, Constructivism in Mathematics, volume I, Studies in Logic and the Foundations of Mathematics 121, Elsevier, (1988).

Vi sono molti modi di tradurre la logica classica in quella intuizionista e viceversa. Un compendio si può trovare in *A.S. Troelstra* e *H. Schwichtenberg*, Basic Proof Theory, Cambridge Tracts in Theoretical Computer Science 43, Cambridge University Press, (1996).

Logica

Lezione 19

Dr Marco Benini

marco.benini@uninsubria.it

Dipartimento di Scienza e Alta Tecnologia
Università degli Studi dell'Insubria

a.a. 2020/21

Syllabus

Logica proposizionale intuizionista

- Isomorfismo Curry-Howard
- Dimostrazioni come programmi
- Correttezza per costruzione
- Normalizzazione forte
- Terminazione di programmi

Isomorfismo Curry-Howard

Se poniamo fianco a fianco le formule della logica proposizionale intuizionista e i tipi della teoria dei tipi semplici, otteniamo

tipi	formule
variabile	variabile
0	$\bot$
1	$\top$
$\alpha \times \beta$	$\alpha \wedge \beta$
$\alpha + \beta$	$\alpha \vee \beta$
$\alpha \to \beta$	$\alpha \supset \beta$

Questa corrispondenza mostra che possiamo tradurre ogni formula logica in un tipo e ogni tipo in una formula mediante una mappa biunivoca.

Isomorfismo Curry-Howard

Se poniamo fianco a fianco le prove proposizionali intuizioniste in deduzione naturale, e i termini della teoria dei tipi semplici, otteniamo:

prova	assunzione	$\top I$	$\bot E$	$\wedge I$	$\wedge E_{1,2}$	$\vee I_{1,2}$	$\vee E$	$\supset I$	$\supset E$
termine	variabile	$*$	$\square_\alpha$	$\langle \_, \_ \rangle$	π_1, π_2	i_1^α, i_2^α	δ	λ	$\cdot$

C'è una evidente corrispondenza biunivoca, che fa perfettamente il paio con quella sui tipi e le formule.

Questa doppia corrispondenza è detta *isomorfismo di Curry-Howard* o anche *interpretazione delle proposizioni come tipi.*

Isomorfismo Curry-Howard

Vediamo un paio di esempi:

- se $A\colon \beta$ è un termine e $x\colon \alpha$ è una variabile, allora $\lambda x\colon \alpha.A\colon \alpha \to \beta$ diviene

$$\dfrac{\begin{array}{c}[\alpha]^{*}\\ \vdots\\ \beta\end{array}}{\alpha \supset \beta}\supset \mathrm{I}^{*}$$

  dove l'etichetta $*$ sta per x.

- se $A\colon \alpha$ e $B\colon \beta$ sono termini, $\langle A, B\rangle\colon \alpha \times \beta$ diventa

$$\dfrac{\begin{array}{cc}\vdots & \vdots\\ A & B\\ \alpha & \beta\end{array}}{\alpha \wedge \beta}\wedge \mathrm{I}$$

Isomorfismo Curry-Howard

A prima vista, secondo l'isomorfismo, la teoria dei tipi semplici è soltanto un modo alternativo per scrivere formule e prove, invece di adottare la notazione ad alberi della deduzione naturale.

Ma la sintassi logica è accoppiata ad una semantica, e la teoria dei tipi ad un meccanismo di computazione dato dalle regole di riduzione.

Isomorfismo Curry-Howard

L'isomorfismo di Curry-Howard può essere esteso alla teoria dei tipi dipendenti. E la controparte diviene un sistema logico che corrisponde, a grandi linee, alla logica intuizionista all'ordine superiore.

In realtà, con l'aggiunta di un paio di assiomi, questa estensione può essere ampliata e approfondita. Infatti, oltre a porre in corrispondenza formule e tipi, e prove e programmi, i tipi possono anche essere interpretati come spazi, e i programmi con punti di quegli spazi.

Inoltre, il tipo dell'uguaglianza $a =_A b$, che contiene le dimostrazioni che $a : A$ è uguale a $b : A$, rappresenta il tipo dei percorsi da a a b nello spazio A.

Questa interpretazione estesa è la base della *teoria omotopica dei tipi*, un campo estremamente attivo nel panorama della ricerca contemporanea.

Dimostrazioni come programmi

Poiché ogni prova formale in logica intuizionista corrisponde ad un termine tipato, e ogni termine tipato è anche un λ-termine, ogni dimostrazione è un programma che calcola qualcosa.

Possiamo associare ad ogni dimostrazione un oggetto che è una *evidenza* del suo tipo, o della sua conclusione, se si preferisce. Quindi l'evidenza di $A \wedge B$ è una coppia di evidenze di A e B; l'evidenza per $A \vee B$ è una coppia (w, e), con $w \in \{1, 2\}$ che dice quale disgiunto valga e e è una evidenza per esso; l'evidenza di $A \supset B$ è una funzione che mappa ogni evidenza A in un'evidenza di B.

Queste evidenze sono i risultati intermedi delle computazioni effettuate dal λ-termine associato alla prova. Quindi, in un sistema costruttivo, dimostrare un enunciato è, essenzialmente, equivalente a scrivere un programma che soddisfi la specifica data dalla sua conclusione.

Dimostrazioni come programmi

Poiché i termini tipati sono dimostrazioni, possiamo ridurre le dimostrazioni ad una forma normale. Formalizzare questo processo conduce a provare il Teorema di Normalizzazione: ogni dimostrazione possiede una forma normale.

Come conseguenza, considerando una qualsiasi dimostrazione $\pi : \vdash A \vee B$, essa può essere ridotta ad una prova $\pi' : \vdash A \vee B$ in forma normale, il cui ultimo passo è necessariamente una istanza di $\vee I_1$ o di $\vee I_2$. Quindi la conclusione del penultimo passo della prova è A o B, mostrando che la logica è costruttiva.

In modo analogo, considerando una prova $\pi : \vdash \exists x : s.A$, essa può sempre essere ridotta a $\pi' : \vdash \exists x : s.A$ in forma normale, il cui ultimo passo è una istanza di $\exists I$. Quindi la conclusione del penultimo passo sarà della forma $A[t/x]$ per qualche termine t, fornendo un testimone per l'esistenziale.

Correttezza per costruzione

Vedere i programmi funzionali come dimostrazioni, o viceversa, apre la possibilità di concepire la programmazione in modo differente.

Un'esperienza che accomuna chiunque abbia mai scritto codice in vita sua, è che i programmi commettono errori. Per quanto si cerchi di testare approfonditamente un programma, esso continua quasi sempre a nascondere un malfunzionamento. Le sue procedure, in quel particolarissimo caso a cui non avevamo mai pensato, funzionano dando il risultato errato.

È possibile scrivere una procedura perfettamente priva di errori?
Come si può fare?

Correttezza per costruzione

Per ottenere questi risultati vi sono due vie: verificare che la procedura sia una dimostrazione la cui conclusione implichi la specifica del comportamento desiderato (*verifica di correttezza*).
Oppure possiamo dimostrare la verità della specifica ed eseguire la dimostrazione (*correttezza per costruzione*).

A supporto di entrambe le strade vi sono software, detti *proof assistants* e *logical frameworks*, che consentono di sviluppare dimostrazioni verificandone la correttezza formale, e quindi automatizzando la loro traduzione in codice eseguibile, il quale risulta, per l'appunto, corretto per costruzione.

Correttezza per costruzione

In pratica, sia la verifica di correttezza che lo sviluppo di codice corretto per costruzione è difficile e oneroso.

Entrambe le tecniche richiedono elevata competenza, molto tempo e ingenti risorse fisiche, economiche, e di calcolo.

Quindi sono vie attuabili quando il guadagno nella loro applicazione sia conveniente rispetto al loro costo. Tipicamente, questo avviene per sistemi critici quali il trasporto aereo e ferroviario, le centrali nucleari e impianti industriali ad altissimo rischio, i sistemi di risposta automatica in ambito militare, ma anche i sistemi di transazione economica di ingenti capitali.

Normalizzazione forte

Una proprietà strutturale difficile da dimostrare in un programma è la sua terminazione. In casi semplici, che sono anche i più comuni, questa proprietà risulta ovvia. Ma nel caso di algoritmi complessi, questa proprietà è difficile e, almeno in linea di principio, non è garantito che possa essere dimostrata.

Questa osservazione suggerisce un approccio differente: isolare un insieme ampio di programmi, che copra la maggior parte degli algoritmi che vengono usati nella pratica, e garantire *per struttura* che questi terminano sempre.
La struttura è data dall'algebra dei tipi.

Normalizzazione forte

Consideriamo il sottosistema della Teoria dei Tipi Semplici i cui tipi sono

- variabili di tipo;
- spazi di funzioni $A \to B$;
- prodotti $A \times B$.

ed i cui termini sono variabili, applicazioni $f \cdot t$, astrazioni $\lambda x\colon A.\, t$, coppie $\langle t_1, t_2 \rangle$, e proiezioni $\pi_1 t$, $\pi_2 t$.

La riduzione in un passo $\rhd_1$ è la congruenza generata da

- $\pi_1 \langle a, b \rangle \blacktriangleright a$;
- $\pi_2 \langle a, b \rangle \blacktriangleright b$;
- $\langle \pi_1 a, \pi_2 a \rangle \blacktriangleright a$;
- $(\lambda x\colon A.\, t) \cdot a \blacktriangleright t[a/x]$.

Come al solito $\rhd$ è la chiusura riflessiva e transitiva di $\rhd_1$.

Normalizzazione forte

Definizione 19.1 (Fortemente normalizzabile)

Un termine t è detto essere *fortemente normalizzabile* quando nessuna sequenza di riduzione che parta da t, $t \triangleright t_1 \triangleright \cdots \triangleright t_n$ possa essere estesa indefinitamente.

In termini computazionali, significa che t può sempre essere ridotto ad una forma normale o anche, che t, visto come un programma, termina sempre.

Normalizzazione forte

Teorema 19.2 (Normalizzazione forte)

Ogni termine è fortemente normalizzabile.

Questo risultato, che vale per la Teoria dei Tipi Semplici, può essere esteso, con alcune cautele, alla Teoria dei Tipi Dipendenti.

Terminazione di programmi

Il senso del Teorema di Normalizzazione Forte è semplice: ogni procedura che sia scritta come un programma funzionale espresso nella teoria dei tipi semplici o dei tipi dipendenti, termina sempre.

Questo significa che, se il programma è corretto nel senso che il suo tipo implica la specifica logica che vogliamo che venga soddisfatta, allora il programma è anche garantito terminare per ogni possibile input.

Questa caratteristica garantisce che un linguaggio di programmazione basato su queste teorie dei tipi sia molto interessante perché rende molto più semplice scrivere programmi solidi e privi di errori.

Riferimenti

L'interpretazione delle proposizioni come tipi e il teorema di normalizzazione per la logica sono illustrati in molti testi: questa lezione è stata tratta da *A.S. Troelstra* e *H. Schwichtenberg*, Basic Proof Theory, Cambridge Tracts in Theoretical Computer Science 43, Cambridge University Press, (1996). Una analisi della normalizzazione si può trovare in *S. Negri* e *J. Von Plato*, Structural Proof Theory, Cambridge University Press (2001).

Un testo più orientato verso l'informatica è *B.C. Pierce*, Types and Programming Languages, The MIT Press, (2002). Esso illustra anche come una teoria dei tipi possa essere usata come un linguaggio di programmazione.

La teoria omotopica dei tipi è introdotta in *The Univalent Foundation Program*, Homotopy Type Theory, Institute of Advanced Studies, Princeton, (2013). Il capitolo sulla teoria dei tipi è anche una ottima introduzione ai tipi dipendenti.

La dimostrazione del Teorema di Normalizzazione Forte si trova in *J-Y. Girard, Y. Lafont, P. Taylor*. Proofs and Types, Cambridge University Press (1989).

Logica

Lezione 20

Dr Marco Benini

marco.benini@uninsubria.it

Dipartimento di Scienza e Alta Tecnologia
Università degli Studi dell'Insubria

a.a. 2020/21

Syllabus

Risultati limitativi

- Teorema di compattezza
- Modelli non standard
- Esempi di modelli non standard

Completezza

Sia Γ un insieme di formule al prim'ordine.

Teorema 20.1 (Completezza)

Se ogni modello di Γ *rende vera la formula* A*, allora* $\Gamma \vdash A$*.*

La dimostrazione di questo risultato esula dagli obiettivi del corso. Infatti, essa usa in modo sofisticato la teoria degli insiemi e in particolare l'*Assioma della Scelta*.

Vale la pena sottolineare che la dimostrazione **non** è, e non può essere, costruttiva: sappiamo che la dimostrazione $\Gamma \vdash A$ esiste, ma non abbiamo una procedura per costruirla, a differenza di quanto accadeva nel caso della logica proposizionale.

Compattezza

Una proprietà importante della logica al prim'ordine è la *compattezza*.

Teorema 20.2 (Compattezza)

Per ogni insieme di formule Γ*, se ogni sottoinsieme finito di* Γ *ha un modello, allora anche* Γ *ha un modello.*

La dimostrazione di questo risultato è semplice, come corollario del Teorema 20.1, ma inessenziale ai fini di questo corso, pertanto verrà omessa.

Modello standard

Molte teorie matematiche, come l'aritmetica o l'analisi sono naturalmente interpretate in un modello inteso ben definito, detto *modello standard*.

Il modello standard per l'aritmetica è la struttura che interpreta la segnatura nei naturali come

- l'unica sorta è interpretata nell'insieme dei numeri naturali, denotata da $\mathbb{N}$;
- i simboli di funzione sono interpretati nel numero 0, nella funzione successore, e nella solita somma e moltiplicazione.

In modo analogo, il modello standard dell'analisi reale interpreta l'unica sorta nell'insieme dei numeri reali con le solite operazioni.

Ogni modello $(\mathscr{M}, \sigma)$ è detto essere *standard* quando $\mathscr{M}$ è la struttura come sopra, senza alcuna restrizione sulla valutazione σ delle variabili.

Modello standard

Sebbene possa essere fuorviante, noi adottiamo la notazione standard che usa gli stessi simboli per denotare sia gli elementi formali del linguaggio sia il loro significato inteso. In un modello standard, questa scelta non fa alcuna differenza.

Poiché il significato inteso di queste teorie è caratterizzare i modelli standard, sarebbe desiderabile se questi modelli fossero gli unici della teoria. Sfortunatamente, questo non avviene.

Modello non-standard

Definizione 20.3 (Modello non-standard)

Ogni struttura $\mathcal{N}$ sul linguaggio di una teoria T che non sia isomorfa al modello standard $\mathcal{M}$, ma che sia comunque un modello per T è detta *modello non-standard*.

Nella definizione, un isomorfismo tra strutture $f : \mathcal{N} \to \mathcal{M}$ è

- una funzione invertibile tra gli universi;
- per ogni termine t, $f(\llbracket t \rrbracket_{\mathcal{N}}) = \llbracket t \rrbracket_{\mathcal{M}}$.

Due strutture sono isomorfe se esiste un isomorfismo tra di esse.

Modello non-standard

Se esiste un modello non standard, significa che c'è una struttura $\mathcal{N}$ che rende vera la teoria dell'aritmetica o dell'analisi, ad esempio, ma che interpreta un qualche termine in un elemento *e* nell'universo che non può essere mappato in un numero naturale o reale, rispettivamente.

È importante notare che l'elemento *e* deve essere l'immagine di un termine rispetto alla funzione di interpretazione: per esempio, l'insieme dei numeri reali composto dagli interi non negativi, **non** costituisce un modello non-standard per l'aritmetica, anche se i suoi elementi sono costruiti in un modo molto diverso dai numeri naturali.

Modello non-standard

Enunciato 20.4

Esiste un modello non standard dell'aritmetica.

Dimostrazione. (i)

Sia $S^0(0) = 0$, e $S^{i+1}(0) = S\,S^i(0)$. Evidentemente il termine $S^n(0)$ viene interpretato in n in ogni modello.

Sia $\Sigma_n = \{x \neq S^i(0) : i < n\}$ una collezione di formule per ogni $n \in \mathbb{N}$ con x una variabile fissata, e sia $\Sigma = \bigcup_{n\in\mathbb{N}} \Sigma_n$.

Detta $\mathscr{M}$ la struttura del modello standard,e definendo σ_n in modo tale che $\sigma_n(x) = n$, il modello standard $(\mathscr{M}, \sigma_n)$ rende vera Σ_n con tutti gli assiomi dell'aritmetica.

Pertanto, ogni $\Xi \subset \Sigma$ finito ha un modello, essendo contenuto in Σ_n per qualche n. Quindi, per il Teorema di Compattezza 20.2, Σ ha un modello $(\mathscr{N}, \sigma)$ che rende vera anche la teoria dell'aritmetica. ↪

Modello non-standard

↪ Dimostrazione. (ii)

In questo modello $\sigma(x) \neq n$ per ogni $n \in \mathbb{N}$ poiché $[\![S^n(0)]\!]_{\mathcal{N}} = n$ ma $x \neq S^n(0)$ occorre in Σ, quindi, per definizione di interpretazione, $\sigma(x) \neq [\![S^n(0)]\!]_{\mathcal{N}}$.

Quindi esiste un elemento $k \notin \mathbb{N}$ tale che $\sigma(x) = k$. Ma interpretando x su $\mathcal{M}$ ci porta a qualche $n \in \mathbb{N}$, qualsiasi valutazione delle variabili possiamo scegliere. Quindi, ogni funzione che mappi $\mathcal{N}$ in $\mathcal{M}$ deve essere non invertibile sul termine x.

In conclusione, $(\mathcal{N}, \sigma)$ è un modello dell'aritmetica che non è isomorfo ad alcun modello standard, pertanto esso è non-standard. □

Discussione

L'esistenza di un un modello non-standard per l'aritmetica mostra che questa teoria non descrive **esattamente** i numeri naturali e le loro proprietà che possano essere espresse nel linguaggio. Non esattamente, in questo contesto, significa "non soltanto".

Questo fatto è problematico: significa che il significato inteso della teoria dell'aritmetica non è pienamente conoscibile. E significa, in altro senso, che i numeri naturali sfuggono ad una formalizzazione completa.

Esempi

Enunciato 20.5

Fissiamo un linguaggio al prim'ordine con una sola sorta. Se un insieme di formule chiuse S ha modelli finiti arbitrariamente grandi, allora ha anche un modello infinito.

Dimostrazione.

Definiamo $\tau_n = \exists x_1, \ldots, x_n. \bigwedge_{1 \leq i < j \leq n} x_i \neq x_j$. Chiaramente, τ_n vale in ogni modello il cui universo contenga almeno n elementi distinti .

Consideriamo un qualsiasi sottoinsieme finito $F \subseteq S \cup \{\tau_n : n \in \mathbb{N}\}$. Sia $K = F \cap \{\tau_n : n \in \mathbb{N}\}$. Poiché F è finito, K è finito, quindi $m = \max\{n : \tau_n \in K\}$ è definito (se $K = \emptyset$, fissiamo $m = 0$). Pertanto, dato che S ha modelli finiti arbitrariamente grandi per ipotesi, F deve avere un modello finito più grande di m.

Quindi, per il Teorema 20.2, $S \cup \{\tau_n : n \in \mathbb{N}\}$ ha un modello $\mathscr{M}$. Poiché τ_n deve valere per ogni $n \in \mathbb{N}$, $\mathscr{M}$ deve avere più di n elementi distinti nel suo universo per ogni $n \in \mathbb{N}$, quindi deve essere infinito. □

Esempi

La segnatura della teoria dei numeri reali ha una sola sorta, la costante 0, le operazioni di somma e moltiplicazione, e le relazioni di uguaglianza e $<$.

La teoria dei numeri reali R è la collezione di tutte le formule chiuse vere nel modello standard, in cui la sorta è interpretata nei numeri reali, e i cui simboli sono interpretati come uno si attende.

Esempi

Esempio 20.6

Esiste un modello di R in cui sono presenti dei numeri infiniti.

Estendiamo la segnatura con una nuova costante ∞. Sia $T = \{n < \infty : n \in \mathbb{N}\}$ e consideriamo la teoria $R \cup T$.

Se $F \subseteq R \cup T$ è finito, allora il massimo m tale che o $(m < \infty) \in F$ oppure $m = 0$ è definito. Quindi, interpretare ∞ in $m + 1$ nel modello standard dei reali rende F vero.

Quindi, per il Teorema di Compattezza 20.2, $R \cup T$ ha un modello, e ∞ deve essere interpretato in un elemento che sia più grande di ogni numero naturale. Lo stesso modello rende vero R che quindi diviene un modello alternativo dei numeri reali, in cui esiste un elemento infinito.

Esempi

Esempio 20.7

Esiste un modello dei numeri reali in cui esistono degli infinitesimi.

Estendiamo la segnatura con una nuova costante k.
Sia $T = \{0 < k < \frac{1}{n+1} : n \in \mathbb{N}\}$.

Se $F \subseteq R \cup T$ è finito, esiste il massimo m per cui $\left(0 < k < \frac{1}{m+1}\right) \in F$ oppure $m = 0$. Quindi, interpretando k in $\frac{1}{m+2}$, F è valido nel modello standard.

Pertanto, per il Teorema di Compattezza 20.2, $R \cup T$ ha un modello, e k deve essere un infinitesimo. Lo stesso modello valida R, quindi è un modello alternativo dei reali con un elemento infinitesimo.

Riferimenti

L'esistenza di modelli non-standard può essere mostrata in molti modi differenti. Il contenuto di questa lezione è stato adattato da *John Bell* e *Moshé Machover*, A Course in Mathematical Logic, North-Holland, (1977).

Logica

Lezione 21

Dr Marco Benini

marco.benini@uninsubria.it

Dipartimento di Scienza e Alta Tecnologia
Università degli Studi dell'Insubria

a.a. 2020/21

Syllabus

Risultati limitativi

- Aritmetica di Peano
- Rappresentabilità
- Codifica

Aritmetica di Peano

L'aritmetica di Peano è la teoria formale standard per descrivere i numeri naturali e le loro proprietà.

È composta da una serie di assiomi, divisi in gruppi, ed è interpretata nella logica classica al prim'ordine.

La stessa teoria, interpretata nella logica intuizionista al prim'ordine è detta aritmetica di Heyting. Nonostante siano sintatticamente identiche, le loro interpretazioni sono molto differenti. Per esempio, nell'aritmetica di Peano è possibile dimostrare che esistono funzioni non calcolabili, mentre ogni funzione che si può provare esistere nell'aritmetica di Heyting è calcolabile, a causa della natura costruttiva della logica.

Aritmetica di Peano

L'aritmetica di Peano è basata sul linguaggio generato dalla segnatura

$$\langle \{\mathbb{N}\}; \{0 : \mathbb{N}, S : \mathbb{N} \to \mathbb{N}, +, \cdot : \mathbb{N} \times \mathbb{N} \to \mathbb{N}\}; \{= : \mathbb{N} \times \mathbb{N}\} \rangle \ .$$

Il primo gruppo di assiomi definisce cosa sia un numero naturale:

$$\forall x, y. S\, x = S\, y \supset x = y \ ; \tag{1}$$

$$\forall x. S\, x \neq 0 \ . \tag{2}$$

L'idea è che i numeri naturali siano gli elementi dell'algebra libera generata da 0 e S. La funzione successore S, dato un numero x, calcola il numero successivo, $x+1$. Quindi i numeri naturali sono scritti nella rappresentazione unaria, e sono naturalmente provvisti di una struttura di ordine totale con minimo.

Aritmetica di Peano

Il secondo gruppo di assiomi definisce la somma e la moltiplicazione:

$$\forall x.0 + x = x \ ; \tag{3}$$

$$\forall x, y.S\,x + y = S(x + y) \ ; \tag{4}$$

$$\forall x.0 \cdot x = 0 \ ; \tag{5}$$

$$\forall x, y.S\,x \cdot y = x \cdot y + y \ . \tag{6}$$

Vale la pena sottolineare la natura induttiva di queste definizioni.

Aritmetica di Peano

Il terzo e ultimo gruppo di assiomi è uno schema: per ogni formula A,

$$A[0/x] \wedge (\forall x.\, A \supset A[S\,x/x]) \supset \forall x.\, A \tag{7}$$

Questo schema formalizza l'induzione sulla struttura dei numeri naturali:

- se A vale per 0
- e, assumendo che A valga su x, possiamo dimostrare che A valga su $S\,x$,
- allora, A vale per ogni $x \in \mathbb{N}$.

Entità rappresentabili

La questione che vogliamo affrontare è come possiamo rappresentare le entità di cui tratta l'aritmetica (numeri, funzioni, relazioni, formule e dimostrazioni) all'interno dell'aritmetica di Peano.

Nel caso dei numeri, è molto semplice:

Definizione 21.1 (Numerali)

Dato $n \in \mathbb{N}$, il *numerale* $\overline{n}$ che rappresenta n è definito come $\overline{0} \equiv 0$, e $\overline{n+1} \equiv S\ \overline{n}$.

Entità rappresentabili

Definizione 21.2 (Rappresentazione)

Una relazione $R \subseteq \mathbb{N}^k$ è *rappresentabile* nell'aritmetica di Peano se e solo se esiste una formula ϕ tale che

- se $(n_1, \ldots, n_k) \in R$ allora $\vdash_{\mathrm{PA}} \phi(\overline{n_1}, \ldots, \overline{n_k})$;
- se $(n_1, \ldots, n_k) \notin R$ allora $\vdash_{\mathrm{PA}} \neg\phi(\overline{n_1}, \ldots, \overline{n_k})$;

dove $\vdash_{PA}$ significa 'dimostrabile nell'aritmetica di Peano'.

Una funzione $f : \mathbb{N}^k \to \mathbb{N}$ è *rappresentabile* nell'aritmetica di Peano se la relazione $R = \{(n_1, \ldots, n_k, m) : m = f(n_1, \ldots, n_k)\}$ è rappresentabile.

Un insieme $S \subset \mathbb{N}$ è *rappresentabile* nell'aritmetica di Peano se la sua funzione caratteristica è rappresentabile.

Entità rappresentabili

Enunciato 21.3

Se le relazioni $P, Q \subseteq \mathbb{N}^k$ sono rappresentabili nell'aritmetica di Peano, lo sono anche $\neg P$, $P \wedge Q$, e $P \vee Q$.

Dimostrazione.

Poiché P e Q sono rappresentabili, ci sono ϕ_P e ϕ_Q come per la Definizione 21.2.

Quindi, $(n_1, \ldots, n_k) \in \neg P$ se e solo se $(n_1, \ldots, n_k) \notin P$. Perciò, $\neg\phi_P$ rappresenta $\neg P$, perché $\neg\neg\phi_P(n_1, \ldots, n_k) = \phi_P(n_1, \ldots, n_k)$.

Anche, $(n_1, \ldots, n_k) \in P \wedge Q$ se e solo se $(n_1, \ldots, n_k) \in P$ e $(n_1, \ldots, n_k) \in Q$.

Quindi $\phi_{P \wedge Q} = \phi_P \wedge \phi_Q$. In modo simile, $\phi_{P \vee Q} = \phi_P \vee \phi_Q$. □

Entità rappresentabili

Teorema 21.4

Tutte le funzioni calcolabili sono rappresentabili nell'aritmetica di Peano.

Corollario 21.5

Tutti gli insiemi ricorsivi e le relazioni ricorsive sono rappresentabili nell'aritmetica di Peano.

È importante notare che la dimostrazione, per quanto complessa, è costruttiva: data una funzione calcolabile f, la dimostrazione fornisce un metodo per costruire la formula che rappresenta f.

Codifica dei termini

Nel seguito, per semplicità, assumeremo che l'insieme delle variabili nel linguaggio dell'aritmetica di Peano sia $V = \{x_i : i \in \mathbb{N}\}$.

Definizione 21.6 (Codifica dei termini)

La *funzione di codifica di Gödel* g sui termini è induttivamente definita come:

- $g(0) = 2 \cdot 3$;
- $g(x_i) = 2 \cdot 3^2 \cdot 5^{i+1}$;
- $g(S\,t) = 2 \cdot 3^3 \cdot 5^{g(t)}$;
- $g(t + s) = 2 \cdot 3^4 \cdot 5^{g(t)} \cdot 7^{g(s)}$;
- $g(t \cdot s) = 2 \cdot 3^5 \cdot 5^{g(t)} \cdot 7^{g(s)}$.

Codifica dei termini

Grazie al teorema che afferma che i numeri naturali ammettono un'unica fattorizzazione in numeri primi, g è calcolabile, iniettiva e g^{-1} è calcolabile.

Qualche commento è necessario:

- ogni codice di un termine è della forma $2 \cdot n$, con n dispari;
- l'esponente del fattore 3 dice se un termine è 0, una variabile, un successore, una somma o una moltiplicazione;
- i parametri di un termine, ovvero l'indice della variabile, o l'argomento del successore, o gli argomenti della somma o della moltiplicazione sono gli esponenti dei fattori 5 e 7, nell'ordine.

Quindi, intuitivamente è possibile scrivere una formula nell'aritmetica di Peano che vale se e solo se il suo argomento è il codice di un termine. Questo può essere formalizzato mostrando che l'insieme dei codici dei termini è ricorsivo, in modo tale che l'Enunciato 21.5 fornisca il risultato desiderato.

Codifica delle formule

Definizione 21.7 (Codifica delle formule)

La *funzione di codifica di Gödel* g sulle formule estende la codifica sui termini ed è induttivamente definita come segue:

- $g(\top) = 2^2 \cdot 3$;
- $g(\bot) = 2^2 \cdot 3^2$;
- $g(t = s) = 2^2 \cdot 3^3 \cdot 5^{g(t)} \cdot 7^{g(s)}$;
- $g(\neg A) = 2^2 \cdot 3^4 \cdot 5^{g(A)}$;
- $g(A \wedge B) = 2^2 \cdot 3^5 \cdot 5^{g(A)} \cdot 7^{g(B)}$;
- $g(A \vee B) = 2^2 \cdot 3^6 \cdot 5^{g(A)} \cdot 7^{g(B)}$;
- $g(A \supset B) = 2^2 \cdot 3^7 \cdot 5^{g(A)} \cdot 7^{g(B)}$;
- $g(\forall x.A) = 2^2 \cdot 3^8 \cdot 5^{g(A)} \cdot 7^{g(x)}$;
- $g(\exists x.A) = 2^2 \cdot 3^9 \cdot 5^{g(A)} \cdot 7^{g(x)}$.

Di nuovo, la codifica g è calcolabile, iniettiva, e g^{-1} è calcolabile.

Codifica delle formule

- ogni codice di una formula è della forma $2^2 \cdot n$, con n dispari, quindi possiamo separare i codici dei termini da quelli delle formule semplicemente guardando l'esponente del fattore 2;
- l'esponente del fattore 3 dice che tipo di formula il codice rappresenta;
- i parametri della formula sono gli esponenti dei fattori 5 e 7, nell'ordine.

Quindi, intuitivamente, è possibile scrivere una formula nell'aritmetica di Peano che dice se il suo argomento sia il codice di una formula. Questo fatto può essere formalizzato mostrando che l'insieme dei codici delle formule è ricorsivo e, in tal modo, l'Enunciato 21.5 fornisce il risultato.

Codifica delle sequenze

Definizione 21.8 (Codifica delle sequenze finite)

La *funzione di codifica di Gödel* g per le sequenze finite $n_1, \ldots, n_k$ di numeri naturali è $g(n_1, \ldots, n_k) = 2^3 \cdot \prod_{1 \leq i \leq k} p_{i+1}^{n_i+1}$, con p_j il j-esimo numero primo.

Chiaramente, la funzione di codifica è computabile, iniettiva e la sua inversa è calcolabile. Anche, i codici delle sequenze sono separati dai codici delle formule e dei temini, e l'insieme dei codici delle sequenze può essere rappresentato, nel senso dell'Enunciato 21.5, da qualche formula nell'aritmetica di Peano.

Codifica delle prove

Definizione 21.9 (Codifica delle prove)

La *funzione di codifica di Gödel* g per le prove estende le precedenti, ed è definita induttivamente come:

- $g\left(\dfrac{\pi_1 : \Gamma \vdash A \quad \pi_2 : \Gamma \vdash B}{A \wedge B}\,\wedge\mathrm{I}\right) = 2^4 \cdot 3 \cdot 5^{g(\pi_1 : \Gamma \vdash A)} \cdot 7^{g(\pi_2 : \Gamma \vdash B)} \cdot 13^{g(A \wedge B)}$;
- $g\left(\dfrac{\pi : \Gamma \vdash A \wedge B}{A}\,\wedge\mathrm{E}_1\right) = 2^4 \cdot 3^2 \cdot 5^{g(\pi : \Gamma \vdash A \wedge B)} \cdot 13^{g(A)}$;
- $g\left(\dfrac{\pi : \Gamma \vdash A \wedge B}{B}\,\wedge\mathrm{E}_2\right) = 2^4 \cdot 3^3 \cdot 5^{g(\pi : \Gamma \vdash A \wedge B)} \cdot 13^{g(B)}$;
- $g\left(\dfrac{\pi : \Gamma \vdash A}{A \vee B}\,\vee\mathrm{I}_1\right) = 2^4 \cdot 3^4 \cdot 5^{g(\pi : \Gamma \vdash A)} \cdot 13^{g(A \vee B)}$;
- $g\left(\dfrac{\pi : \Gamma \vdash B}{A \vee B}\,\vee\mathrm{I}_2\right) = 2^4 \cdot 3^5 \cdot 5^{g(\pi : \Gamma \vdash B)} \cdot 13^{g(A \vee B)}$;

↪

Codifica delle prove

↪ (Codifica delle prove)

- $g\left(\dfrac{\pi_1 : \Gamma \vdash A \vee B \quad \pi_2 : \Gamma, A \vdash C \quad \pi_3 : \Gamma, B \vdash C}{C}\,\vee\mathsf{E}\right) =$
  $2^4 \cdot 3^6 \cdot 5^{g(\pi_1 : \Gamma \vdash A \vee B)} \cdot 7^{g(\pi_2 : \Gamma, A \vdash C)} \cdot 11^{g(\pi_3 : \Gamma, B \vdash C)} \cdot 13^{g(C)}$;
- $g\left(\dfrac{\pi : \Gamma, A \vdash B}{A \supset B}\,\supset\mathsf{I}\right) = 2^4 \cdot 3^7 \cdot 5^{g(\pi : \Gamma, A \vdash B)} \cdot 13^{g(A \supset B)}$;
- $g\left(\dfrac{\pi_1 : \Gamma \vdash A \supset B \quad \pi_2 : \Gamma \vdash A}{B}\,\supset\mathsf{E}\right) = 2^4 \cdot 3^8 \cdot 5^{g(\pi_1 : \Gamma \vdash A \supset B)} \cdot 7^{g(\pi_2 : \Gamma \vdash A)} \cdot 13^{g(B)}$;
- $g\left(\dfrac{\pi : \Gamma, A \vdash \bot}{\neg A}\,\neg\mathsf{I}\right) = 2^4 \cdot 3^9 \cdot 5^{g(\pi : \Gamma, A \vdash \bot)} \cdot 13^{g(\neg A)}$;
- $g\left(\dfrac{\pi_1 : \Gamma \vdash \neg A \quad \pi_2 : \Gamma \vdash A}{\bot}\,\neg\mathsf{E}\right) = 2^4 \cdot 3^{10} \cdot 5^{g(\pi_1 : \Gamma \vdash \neg A)} \cdot 7^{g(\pi_2 : \Gamma \vdash A)} \cdot 13^{g(\bot)}$;
- $g\left(\dfrac{}{\top}\,\top\mathsf{I}\right) = 2^4 \cdot 3^{11} \cdot 13^{g(\top)}$;

↪

Codifica delle prove

↪ (Codifica delle prove)

- $g\left(\dfrac{\pi : \Gamma \vdash \bot}{A}\,\bot\mathsf{E}\right) = 2^4 \cdot 3^{12} \cdot 5^{g(\pi : \Gamma \vdash \bot)} \cdot 13^{g(A)}$;
- $g\left(\dfrac{}{A \vee \neg A}\,\mathsf{lem}\right) = 2^4 \cdot 3^{13} \cdot 13^{g(A \vee \neg A)}$;
- $g\left(\dfrac{\pi : \Gamma \vdash A}{\forall x.A}\,\forall\mathsf{I}\right) = 2^4 \cdot 3^{14} \cdot 5^{g(\pi : \Gamma \vdash A)} \cdot 13^{g(\forall x.A)} \cdot 19^{g(x)}$;
- $g\left(\dfrac{\pi : \Gamma \vdash \forall x.A}{A[t/x]}\,\forall\mathsf{E}\right) = 2^4 \cdot 3^{15} \cdot 5^{g(\pi : \Gamma \vdash \forall x.A)} \cdot 13^{g(A[t/x])} \cdot 17^{g(t)} \cdot 19^{g(x)}$;
- $g\left(\dfrac{\pi : \Gamma \vdash A[t/x]}{\exists x.A}\,\exists\mathsf{I}\right) = 2^4 \cdot 3^{16} \cdot 5^{g(\pi : \Gamma \vdash A[t/x])} \cdot 13^{g(\exists x.A)} \cdot 17^{g(t)} \cdot 19^{g(x)}$;
- $g\left(\dfrac{\pi_1 : \Gamma \vdash \exists x.A \quad \pi_2 : \Gamma, A \vdash B}{B}\,\exists\mathsf{E}\right) =$
  $2^4 \cdot 3^{17} \cdot 5^{g(\pi_1 : \Gamma \vdash \exists x.A)} \cdot 7^{g(\pi_2 : \Gamma, A \vdash B)} \cdot 13^{g(B)} \cdot 19^{g(x)}$;

↪

Codifica delle prove

↪ (Codifica delle prove)

- $g\left(\dfrac{}{\forall x.x=x}\,\mathsf{ax}\right) = 2^4 \cdot 3^{18} \cdot 13^{g(\forall x.x=x)} \cdot 19^{g(x)}$;
- $g\left(\dfrac{}{\forall x,y.x=y\supset y=x}\,\mathsf{ax}\right) = 2^4 \cdot 3^{19} \cdot 13^{g(\forall x,y.x=y\supset y=x)} \cdot 19^{g(x,y)}$;
- $g\left(\dfrac{}{\forall x,y,z.x=y\wedge y=z\supset x=z}\,\mathsf{ax}\right) =$
  $2^4 \cdot 3^{20} \cdot 13^{g(\forall x,y,z.x=y\wedge y=z\supset x=z)} \cdot 19^{g(x,y,z)}$;
- $g\left(\dfrac{\pi_1:\Gamma\vdash A[t/x] \quad \pi_2:\Gamma\vdash t=r}{A[r/x]}\,\mathsf{ax}\right) =$
  $2^4 \cdot 3^{21} \cdot 5^{g(\pi_1:\Gamma\vdash A[t/x])} \cdot 7^{g(\pi_2:\Gamma\vdash t=r)} \cdot 13^{g(A[r/x])} \cdot 19^{g(x)}$;
- $g\left(\dfrac{}{\forall x_1,\ldots,x_n.\exists!z.z=f(x_1,\ldots,x_n)}\,\mathsf{ax}\right) =$
  $2^4 \cdot 3^{22} \cdot 13^{g(\forall x_1,\ldots,x_n.\exists!z.z=f(x_1,\ldots,x_n))} \cdot 17^{g(f(x_1,\ldots,x_n))} \cdot 19^{g(x_1,\ldots,x_n,z)}$;
- $g\left(\dfrac{}{\forall x.Sx\neq x}\,\mathsf{ax}\right) = 2^4 \cdot 3^{23} \cdot 13^{g(\forall x.Sx\neq x)} \cdot 19^{g(x)}$;

↪

Codifica delle prove

↪ (Codifica delle prove)

- $g\left(\dfrac{}{\forall x,y.\,Sx=Sy\supset x=y}\,\text{ax}\right)=2^4\cdot 3^{24}\cdot 13^{g(\forall x,y.\,Sx=Sy\supset x=y)}\cdot 19^{g(x,y)}$;
- $g\left(\dfrac{}{\forall x.\,0+x=x}\,\text{ax}\right)=2^4\cdot 3^{25}\cdot 13^{g(\forall x.\,0+x=x)}\cdot 19^{g(x)}$;
- $g\left(\dfrac{}{\forall x,y.\,Sx+y=S(x+y)}\,\text{ax}\right)=2^4\cdot 3^{26}\cdot 13^{g(\forall x,y.\,Sx+y=S(x+y))}\cdot 19^{g(x,y)}$;
- $g\left(\dfrac{}{\forall x.\,0\cdot x=0}\,\text{ax}\right)=2^4\cdot 3^{27}\cdot 13^{g(\forall x.\,0\cdot x=0)}\cdot 19^{g(x)}$;
- $g\left(\dfrac{}{\forall x,y.\,Sx\cdot y=x\cdot y+y)}\,\text{ax}\right)=2^4\cdot 3^{28}\cdot 13^{g(\forall x,y.\,Sx\cdot y=x\cdot y+y))}\cdot 19^{g(x,y)}$;
- $g\left(\dfrac{}{A[0/x]\wedge(\forall x.\,A\supset A[Sx/x])\supset\forall x.\,A}\,\text{ax}\right)=$
  $2^4\cdot 3^{29}\cdot 5^{g(A)}\cdot 13^{g(A[0/x]\wedge(\forall x.\,A\supset A[Sx/x])\supset\forall x.\,A)}\cdot 19^{g(x)}$;
- se $A\in\Gamma$ è una prova per assunzione, $g(A)=2^4\cdot 3^{30}\cdot 5^{g(A)}\cdot 7^{g(\Gamma)}\cdot 13^{g(A)}$ con $\Gamma=\{\gamma_1,\ldots,\gamma_n\}$ e $g(\Gamma)=g(\gamma_1,\ldots,\gamma_n)$.

↪

Codifica delle prove

↪ (Codifica delle prove)

È importante rimarcare che $g(e_1, \ldots, e_n)$, quando e_i non sono numeri, deve essere letto come $g(g(e_1), \ldots, g(e_n))$, il codice della sequenza dei codici degli elementi.

Sebbene sia lungo e tedioso da verificare, g è iniettiva, computabile e g^{-1} è calcolabile. Anche, la funzione di codifica è scritta per rendere semplice estrarre le componenti. Ad esempio, il codice della conclusione è sempre l'esponente del fattore 13.

Numerali

Definizione 21.10 (Numerale)

Il *numerale* $\ulcorner A \urcorner$ di una formula A è definito come $\ulcorner A \urcorner = S^{g(A)}(0)$, il codice di A scritto nella sintassi dell'aritmetica di Peano.

In modo analogo, il numerale di un termine t è $\ulcorner t \urcorner = S^{g(t)}(0)$, il numerale di una dimostrazione π è $\ulcorner \pi \urcorner = S^{g(\pi)}(0)$, e il numerale di una sequenza finita è $\ulcorner e_1, \ldots, e_n \urcorner = S^{g(e_1,\ldots,e_n)}(0)$.

I numerali permettono di *internalizzare* i codici: possiamo, in modo indiretto, parlare di una formula (termine, dimostrazione, sequenza) enunciando la proprietà del suo codice. Fino a che questa proprietà non dipende dal valore, ma dal 'significato' del codice, è un modo perfettamente ragionevole di procedere.

Riferimenti

L'aritmetica di Peano è descritta in tutti i testi universitari di logica. La presentazione in questo corso è tratta da *John Bell* e *Moshé Machover*, A Course in Mathematical Logic, North-Holland, (1977).

La nozione di rappresentabilità di relazioni, insiemi e funzioni segue *Barry Cooper*, Computability Theory, Chapman & Hall/CRC Mathematics, (2004). La dimostrazione del Teorema 21.4 si può trovare in *Elliott Mendelson*, Introduction to Mathematical Logic, CRC Press.

La codifica di termini, formule e dimostrazioni è tratta da *John Bell* e *Moshé Machover*, A Course in Mathematical Logic, North-Holland, (1977).

Logica

Lezione 22

Dr Marco Benini

marco.benini@uninsubria.it

Dipartimento di Scienza e Alta Tecnologia
Università degli Studi dell'Insubria

a.a. 2020/21

Syllabus

Risultati limitativi

- Lemma di punto fisso
- Primo teorema di incompletezza di Gödel

Incompletezza

Lo scopo di questa lezione è dimostrare il teorema di incompletezza di Gödel.

Questo teorema afferma che esiste una formula G chiusa nell'aritmetica di Peano, e in ogni teoria simile, che sia vera nel modello standard, ma non dimostrabile.

Lemma di punto fisso

Lemma 22.1 (Punto fisso)

Sia Ξ una teoria in cui ogni funzione calcolabile sia rappresentabile, e sia A una formula tale che $\mathrm{FV}(A) = \{y\}$.
Allora, esiste una formula δ_A tale che $\mathrm{FV}(\delta_A) = \emptyset$ *e* $\vdash_\Xi \delta_A = A[\ulcorner \delta_A \urcorner / y]$.

Dimostrazione. (i)

Si può facilmente dimostrare nella logica pura che

$$\vdash B[k/z] = (\exists z.\, z = k \wedge B)$$

per ogni formula B e per ogni termine k della stessa sorta di z.

Sia $\Delta_{\mathscr{F}}$ la mappa da formule a formule definita come

$$\Delta_{\mathscr{F}}(B) \equiv \exists z.\, z = \ulcorner B \urcorner \wedge B \ .$$

Questa funzione è evidentemente calcolabile. ↪

Lemma di punto fisso

↪ Dimostrazione. (ii)

Quindi la mappa $\Delta_{\mathbb{N}}$ definita da

$$\Delta_{\mathbb{N}}(g(B)) = g(\Delta_{\mathscr{F}}(B))$$

è totale sull'immagine di g e calcolabile.

Per ipotesi, esiste una formula Δ con $\mathrm{FV}(\Delta) = \{x, y\}$ tale che Δ rappresenta la funzione $\Delta_{\mathbb{N}}$. In particolare, è dimostrabile che

$$\vdash_{\Xi} (y = \ulcorner \Delta_{\mathbb{N}}(g(B)) \urcorner) = \Delta[\ulcorner B \urcorner / x]$$

Senza ledere la generalità, possiamo definire

$$\delta_A \equiv \Delta_{\mathscr{F}}(F)$$

per una qualche formula F da determinarsi. ↪

Lemma di punto fisso

↪ Dimostrazione. (iii)

$$
\begin{aligned}
& A[\ulcorner \delta_A \urcorner / y] \\
\equiv\ & A[\ulcorner \Delta_{\mathscr{F}}(F) \urcorner / y] && \text{(definizione di } \delta_A\text{)} \\
\equiv\ & A[\ulcorner \Delta_{\mathbb{N}}(g(F)) \urcorner / y] && \text{(definizione di } \Delta_{\mathbb{N}}\text{)} \\
=\ & \exists y.\, y = \ulcorner \Delta_{\mathbb{N}}(g(F)) \urcorner \wedge A && \text{(evitando la sostituzione)} \\
=\ & \exists y.\, \Delta[\ulcorner F \urcorner / x] \wedge A && \text{(definizione di } \Delta\text{)} \\
=\ & \exists x.\, x = \ulcorner F \urcorner \wedge \exists y.\, \Delta \wedge A && \text{(evitando la sostituzione)}
\end{aligned}
$$

Quindi, ponendo $F \equiv \exists y.\, \Delta \wedge A$,

$$
\begin{aligned}
\equiv\ & \exists x.\, x = \ulcorner F \urcorner \wedge F && \text{(definizione di } F\text{)} \\
\equiv\ & \Delta_{\mathscr{F}}(F) && \text{(definizione di } \Delta_{\mathscr{F}}\text{)} \\
\equiv\ & \delta_A && \text{(definizione di } \delta_A\text{)}
\end{aligned}
$$

□

Predicato di provabilità

Definizione 22.2 (Predicato di provabilità)

La formula $\mathscr{D}$ con $\mathrm{FV}(\mathscr{D}) = \{x, y\}$ è definita come
$\mathscr{D} \equiv \exists z.\, 13^y \cdot z = x \wedge \mathrm{isExpr}(x) \wedge \mathrm{isExpr}(y) \wedge \mathrm{isProof}(x) \wedge \mathrm{isFormula}(y)$.

Il *predicato di provabilità* T è la formula $\exists x.\mathscr{D}$, con $FV(T) = \{y\}$.

Chiaramente, $\mathscr{D}[\ulcorner \pi \urcorner / x, \ulcorner A \urcorner / y]$ vale esattamente quando A è la conclusione della dimostrazione $\pi \colon \vdash A$. E, di conseguenza, $T[\ulcorner A \urcorner / y]$ vale quando A è dimostrabile.

Le formule isExpr(_), isProof(_), e isFormula(_) nella definizione di $\mathscr{D}$ non sono state rese esplicite in quanto derivano dalla lezione precedente.

Teorema di incompletezza

Teorema 22.3 (Primo teorema di incompletezza di Gödel)

Sia T una teoria effettiva che sia consistente e in grado di rappresentare le funzioni calcolabili. Allora, esiste una formula chiusa G tale che $T \nvdash G$ e $T \nvdash \neg G$.

Dimostrazione.

Consideriamo la formula $\neg T[x/y]$: applicando il Lemma di Punto Fisso, esiste G tale che $\mathrm{FV}(G) = \emptyset$ e $\vdash G = \neg T[\ulcorner G \urcorner / y]$.

Assumiamo che vi sia $\pi : \vdash G$. Allora $\vdash \neg T[\ulcorner G \urcorner / y]$. Ma, poiché $\pi : \vdash G$, vale che $\vdash \mathscr{D}[\ulcorner \pi \urcorner / x, \ulcorner G \urcorner / y]$, e quindi $\vdash \exists x.\mathscr{D}[\ulcorner G \urcorner / y]$, ovvero $\vdash T[\ulcorner G \urcorner / y]$, rendendo la teoria non consistente. Quindi $\nvdash G$.

D'altro canto, supponiamo esista $\pi : \vdash \neg G$. Allora $\vdash T[\ulcorner G \urcorner / y]$ per definizione di G, quindi $\vdash \exists x.\mathscr{D}[\ulcorner G \urcorner / y]$. Ma questo implica che esiste $\theta : \vdash G$ con $x = \ulcorner \theta \urcorner$. Quindi, di nuovo, otteniamo una contraddizione. Pertanto $\nvdash \neg G$. □

Riferimenti

La dimostrazione originale del primo teorema di incompletezza si può trovare in *Kurt Gödel*, Über formal unentscheidbare Sätze der Principia Mathematica und verwandter Systeme, I, Monatshefte für Mathematik und Physik 38, 173–198, (1931).

La prova è stata generalizzata e ripulita da Rosser, e noi abbiamo presentato una versione lievemente ritoccata del risultato di Rosser. Il riferimento è *John Barkley Rosser*, Extensions of some theorems of Gödel and Church, Journal of Symbolic Logic 1, 87–91 (1936).

Un resoconto complessivo si può trovare in *John Bell* e *Moshé Machover*, A Course in Mathematical Logic, North-Holland (1977). Tuttavia, questa lezione è stata preparata seguendo a grandi linee le note non pubblicate del corso tenuto da Silvio Valentini nel 1991.

Logica

Lezione 23

Dr Marco Benini

marco.benini@uninsubria.it

Dipartimento di Scienza e Alta Tecnologia
Università degli Studi dell'Insubria

a.a. 2020/21

Syllabus

Risultati limitativi

- Funzioni calcolabili
- Tesi di Church-Turing
- Teorema di Cantor
- Esistenza di funzioni non calcolabili

Funzioni calcolabili

Una funzione $f : \mathbb{N} \to \mathbb{N}$ è, intuitivamente, *calcolabile* se esiste un programma per computer che, dato $n \in \mathbb{N}$ in input, produca in output $f(n)$ dopo un tempo finito.

Questa definizione è insoddisfacente per numerosi motivi:

- Fissa una macchina, il computer, come riferimento. Se usassimo un altro strumento meccanico, saremmo in grado di calcolare funzioni non computabili da un computer?
- Non consente di discernere le funzioni calcolabili da quelle che non lo sono. Data una funzione, è possibile scrivere un programma che la calcoli?
- Non mostra alcun metodo per esibire una funzione non calcolabile. Esiste una funzione che non si possa calcolare? Abbiamo un esempio?

Funzioni calcolabili

Il λ-calcolo, le macchine di Turing, le funzioni ricorsive parziali di Kleene, il modello di Von Neumann sono esempi di sistemi di calcolo, oltre a numerosissimi altri.

Essi sono *equivalenti* nel senso che ogni funzione che possa essere calcolata in uno di essi, può essere computata in qualsiasi altro.

Sebbene siano paradigmi di calcolo molto differenti, essi presentano una caratteristica comune importante: una *istanza universale*. Un oggetto che computa, il quale, preso in input un codice e un valore, calcola la funzione corrispondente a quel codice su quel valore.

Tesi di Church-Turing

La *Tesi di Church-Turing* afferma che tutte e sole le funzioni calcolabili sono quelle che possono essere computate in uno di questi sistemi di calcolo.

La tesi è **solida**: per quanto si siano esplorati paradigmi di calcolo molto differenti (dai sistemi biologici ai sistemi quantistici), nessun sistema effettivamente realizzabile è mai risultato in grado di calcolare funzioni che non sono calcolabili nel senso di Church-Turing.

La tesi è **matematicamente fondata**: questi sistemi di calcolo possono essere descritti in modo formale e preciso in termini matematici, e pertanto possono essere efficacemente studiati. E godono di proprietà interessanti basate su una struttura molto solida e largamente indipendente dalle specifiche caratteristiche del paradigma di calcolo.

Tesi di Church-Turing

Storicamente, questi sistemi di calcolo precedono la nascita dell'Informatica. Anzi, la loro definizione e studio è il momento di nascita di questa disciplina.

In particolare, l'istanza universale in questi sistemi corrisponde a quell'oggetto che oggi chiamiamo *computer ideale*, senza le limitazioni fisiche di memoria o risorse di calcolo.

Sebbene la tesi di Church-Turing, per sua natura, non sia dimostrabile, essa è universalmente accettata e costituisce uno dei fondamenti teorici e filosofici dell'Informatica.

Comparare insiemi

Diciamo che due insiemi A e B sono *equivalenti*, $A \cong B$, se esiste una funzione biiettiva tra essi.

Teorema 23.1 (Schröder-Bernstein)

Se $f : A \to B$ è iniettiva e $g : B \to A$ è iniettiva allora $A \cong B$.

Dimostrazione. (i)

Sia $C_0 = A \setminus g(B)$ e, per induzione, $C_{n+1} = \{g(x) : x \in D_n\}$ e $D_n = \{f(x) : x \in C_n\}$. Definiamo

$$h(x) = \begin{cases} f(x) & \text{se } x \in C_n \text{ per qualche } n \\ g^{-1}(x) & \text{altrimenti} \end{cases}$$

Questa definizione è ben fondata poiché $g^{-1}(x)$ è definita su $g(B)$. ↪

Comparare insiemi

↪ Dimostrazione. (ii)

Siano $x, y \in A$. Supponiamo che $h(x) = h(y)$: se $x \in C_m$ e $y \in C_k$ per qualche m e k, allora $f(x) = f(y)$, quindi $x = y$ essendo f iniettiva; se $x \notin C_n$ e $y \notin C_n$ per ogni n, allora $g^{-1}(x) = g^{-1}(y)$, quindi $x = y$ essendo g iniettiva; se $x \in C_m$ per qualche m e $y \notin C_n$ per ogni n, $f(x) = g^{-1}(y)$, quindi $(g \circ f)(x) = y$, ovvero $y \in C_{m+1}$, che è impossibile. Quindi h è iniettiva.

Dobbiamo mostrare che $h(A) = B$. In primo luogo, per ogni n e ogni $z \in D_n$, $z = f(x)$ per qualche $x \in C_n$, quindi per definizione, $z = h(x)$. Poi, sia $z \in B \setminus \bigcup_n D_n$. Evidentemente, per induzione su n, $g(z) \notin C_n$ per ogni n, quindi $h(g(z)) = g^{-1}(g(z)) = z$. Pertanto h è suriettiva. □

Teorema di Cantor

Esempio 23.2

Sia $P = \{2n : n \in \mathbb{N}\}$. Poiché $f : P \to \mathbb{N}$ tale che $f(x) = x$ è iniettiva, e $g : \mathbb{N} \to P$ tale che $g(x) = 2x$ è iniettiva, per il Teorema 23.1 concludiamo che $P \cong \mathbb{N}$.

In generale, un insieme infinito A è tale che sia possibile trovare un sottoinsieme proprio $B \subset A$ tale che $A \cong B$.

Teorema di Cantor

Esempio 23.3
$\mathbb{N} \times \mathbb{N} \cong \mathbb{N}$

Chiaramente, la funzione $f : \mathbb{N} \to \mathbb{N} \times \mathbb{N}$ che mappa $x \mapsto (x, x)$ è iniettiva.
Nel senso opposto, la funzione $g : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ definita come $g(x, y) = (x + y)(x + y + 1)/2 + y$ è iniettiva, come è facile dimostrare.
Informalmente, essa conta le coppie lungo le diagonali, il che giustifica intuitivamente il suo essere iniettiva: la prova formale è soltanto aritmetica.
Quindi, per il Teorema 23.1 il risultato segue.

Una semplice estensione, che si ottiene immediatamente per induzione, è che $\mathbb{N}^k \cong \mathbb{N}$ per ogni $k > 0$.

Teorema di Cantor

Esempio 23.4

La collezione delle sequenze finite di naturali $\mathbb{N}^* \cong \mathbb{N}$

Ovviamente, la funzione $f : \mathbb{N} \to \mathbb{N}^*$ che mappa $x \mapsto \{x\}$ è iniettiva.
Nel senso opposto, chiamando $g_n : \mathbb{N}^n \to \mathbb{N}$ la biiezione dal prodotto cartesiano di $n \geq 1$ copie di $\mathbb{N}$ a $\mathbb{N}$, possiamo definire una funzione $h : \mathbb{N}^* \to \mathbb{N} \times \mathbb{N}$ come $h(\{x_i\}_{1 \leq i \leq n}) = (n, g_n(x_1, \ldots, x_n))$. Per $n = 0$, sia $h(\emptyset) = (0, 0)$.
Evidentemente, h è iniettiva poiché g_n lo è per ogni $n \geq 1$. Quindi, la composizione $g_2 \circ h$ è iniettiva, e il risultato segue dal Teorema 23.1.

Teorema di Cantor

Una applicazione di quanto ottenuto sinora alla logica è immediata: sia Σ una segnatura con un numero finito di simboli. Poiché le variabili di sorta s sono in corrispondenza biiettiva con $\mathbb{N}$, la collezione di tutte le variabili è in corrispondenza biunivoca con $\mathbb{N}$.

Quindi, le sequenze di simboli dati dai simboli di funzione, le parentesi, le virgole e le variabili sono in corrispondenza biunivoca con $\mathbb{N}$. Pertanto, la collezione di tutti i termini su Σ, essendo un sottoinsieme infinito di tale insieme, è in biiezione con $\mathbb{N}$.

Analogamente, la collezione di tutte le formule su Σ, essendo un sottoinsieme infinito della collezione delle sequenze di simboli di Σ più un insieme finito di simboli logici, è in biiezione con $\mathbb{N}$.

Teorema di Cantor

Una applicazione immediata all'Informatica è la seguente: l'insieme di tutti i programmi in uno specifico linguaggio di programmazione è in biiezione con $\mathbb{N}$.

Il risultato consegue dal fatto che l'insieme dei programmi è infinito, ed è contenuto nell'insieme delle sequenze finite di simboli ASCII o ISO.

Teorema di Cantor

Teorema 23.5 (Cantor)

$\wp(\mathbb{N}) \not\cong \mathbb{N}$.

Dimostrazione.

Possiamo identificare ogni sottoinsieme $A \subseteq \mathbb{N}$ con la sua funzione caratteristica $\chi_A : \mathbb{N} \to \{0,1\}$. Supponiamo che tutte queste funzioni siano in biiezione con $\mathbb{N}$: quindi, esiste una funzione biiettiva e che le enumera. Quindi, abbiamo una sequenza $\wp(\mathbb{N}) \cong \{\chi_{A_i}\}_{i \in \mathbb{N}}$ tale che l'i-esima funzione sia data da $e(i)$.
Definiamo la funzione $\Delta : \mathbb{N} \to \{0,1\}$ come $\Delta(x) = 1 - \chi_{A_x}(x)$. Quindi Δ deve apparire da qualche parte nella sequenza, ovvero $\Delta = \chi_{A_k}$ per qualche $k \in \mathbb{N}$. Questo è impossibile poiché $\chi_{A_k}(k) = \Delta(k) = 1 - \chi_{A_k}(k)$ e $\chi_{A_k} \in \{0,1\}$. Quindi le funzioni caratteristiche non sono in biiezione con $\mathbb{N}$, e $\wp(\mathbb{N}) \not\cong \mathbb{N}$. □

Come effetto collaterale, poiché le funzioni $\mathbb{N} \to \{0,1\}$ sono in evidente biiezione con l'intervallo reale $[0,1]$, otteniamo che $\mathbb{R} > \mathbb{N}$ strettamente. In altre parole, l'infinito non è unico!

Esistenza di funzioni non calcolabili

Una applicazione immediata del Teorema 23.5 all'Informatica è l'esistenza di funzioni non calcolabili.

Infatti, come abbiamo visto, l'insieme dei programmi che si possono scrivere in un linguaggio di programmazione è in biiezione con $\mathbb{N}$.

L'insieme delle funzioni $\mathbb{N} \to \{0,1\} \cong [0,1] > \mathbb{N}$. Pertanto, esiste una funzione che non può essere calcolata da un programma. I programmi sono troppo pochi per poter calcolare tutte le funzioni da $\mathbb{N}$ in $\mathbb{N}$, o anche solo quelle $\mathbb{N} \to \{0,1\}$.

Riferimenti

Questa lezione è basata sui risultati di Cantor e sui fondamenti di Informatica Teorica che si possono trovare su qualsiasi testo universitario dedicato.

Logica

Lezione 24

Dr Marco Benini

marco.benini@uninsubria.it

Dipartimento di Scienza e Alta Tecnologia
Università degli Studi dell'Insubria

a.a. 2020/21

Syllabus

Risultati limitativi

- Incalcolabilità e incompletezza
- Esempio: teorema di Kruskal
- Esempio: minori di grafi

Incompletezza e incalcolabilità

La dimostrazione che esistono funzioni non calcolabili è insoddisfacente: essa è corretta, ma non fornisce alcun esempio di funzione che non sia computabile.

Il primo passo che vogliamo compiere, è mostrare una funzione che non possa essere calcolata.

L'esempio che usiamo è il *problema dell'arresto*: dato un programma e un input, ci chiediamo se si possa determinare sempre che tale programma termina su tale input.

Incompletezza e incalcolabilità

Sia $\{\phi_i\}_{i\in\mathbb{N}}$ una enumerazione delle funzioni calcolabili: ogni funzione calcolabile compare nella lista in qualche posizione. Questa assunzione è ragionevole, mediante una opportuna codifica.

Consideriamo la funzione

$$h(i,x) = \begin{cases} 1 & \text{se } \phi_i(x) \text{ termina} \\ 0 & \text{altrimenti} \end{cases} \tag{8}$$

Incompletezza e incalcolabilità

Sia $f : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ una funzione calcolabile totale.
Definiamo

$$g(i) = \begin{cases} 0 & \text{se } f(i,i) = 0 \\ \perp & \text{altrimenti} \end{cases} \tag{9}$$

La funzione parziale g è evidentemente calcolabile.

Quindi, esiste un indice $e \in \mathbb{N}$ tale che $\phi_e = g$.

Incompletezza e incalcolabilità

Consideriamo $f(e,e)$:

- se $f(e,e) = 0$, allora $g(e) = \phi_e(e) = 0$, pertanto $h(e,e) = 1$;
- se $f(e,e) \neq 0$, allora $g(e) = \phi_e(e) = \perp$, ovvero $g(e)$ non termina, quindi $h(e,e) = 0$.

Supponiamo che h sia calcolabile.
Quindi h deve essere totale.
Prendendo $f = h$, abbiamo che $h(e,e) = 1$ se e solo se $h(e,e) = 0$.
Contraddizione! Quindi h non può essere calcolabile.

Incompletezza e incalcolabilità

Teorema 24.1 (Incompletezza debole)

Non esiste una assiomatizzazione effettiva della teoria dei numeri naturali che sia corretta e completa.

Dimostrazione. (i)

Sia $\{\phi_i\}_{i\in\mathbb{N}}$ una enumerazione effettiva di tutte e sole le funzioni calcolabili. In particolare, questo implica che esiste una formula $\mathscr{F}(i,x,y)$ che rappresenta la relazione $\phi_i(x) = y$.

Per assurdo, supponiamo che vi sia una assiomatizzazione effettiva come nell'enunciato. Quindi esiste, per rappresentabilità degli insiemi ricorsivi, una formula $\mathscr{E}(n)$ che enumeri tutte e sole le formule vere nella teoria. ↪

Incompletezza e incalcolabilità

↪ Dimostrazione. (ii)

Consideriamo la formula $\exists y.\mathscr{F}(i,x,y)$: essa è valida se e solo se $\phi_i(x)$ è definita, ovvero se il programma ϕ_i termina su input x.

Quindi possiamo calcolare il più piccolo valore n tale che $\mathscr{E}(n) = \exists y.\mathscr{F}(i,x,y)$ oppure $\mathscr{E}(n) = \neg\exists y.\mathscr{F}(i,x,y)$.

Poiché l'assiomatizzazione è corretta, questa computazione definisce una funzione calcolabile g che, dati i e x, ritorna l'indice della formula che decide se $\phi_i(x)$ termina.

Poiché l'assiomatizzazione è completa, questa funzione è totale.
Quindi esiste una funzione computabile totale che risolve il problema dell'arresto, contraddicendo il risultato precedente.
Pertanto tale assiomatizzazione non può esistere. □

Incompletezza naturale

Una critica che è lecito muovere al primo teorema di incompletezza di Gödel è che esso mostra una formula vera nel modello standard dell'aritmetica di Peano, ma indimostrabile. Tuttavia, questa formula è *artificiale*: non illustra una proprietà genuinamente matematica o attinente all'aritmetica, ma è la codifica di un principio logico. Esula dal dominio proprio della teoria.

Quindi la questione se esista una proprietà matematicamente significativa nel proprio dominio che sia anche indimostrabile e vera, è ancora aperta.

In chiusura del corso, mostriamo due proprietà con queste caratteristiche, le quali hanno rilevanza e applicazioni concrete.

Teorema di Kruskal

Teorema 24.2 (Kruskal)

Esiste un qualche $n \in \mathbb{N}$ tale che, se $T_1, \ldots, T_n$ è una sequenza finita di alberi, dove T_k ha $k+n$ vertici, allora per qualche $i < j$, esiste una mappa iniettiva $f : T_i \to T_j$ tra i vertici che preserva i percorsi.

Far vedere che il teorema di Kruskal non può essere dimostrato nell'aritmetica di Peano richiede una matematica più sofisticata di quella presentata in questo corso. Essenzialmente, ogni funzione che si può dimostrare esistere nell'aritmetica di Peano ha una velocità di crescita che non può superare un certo limite, mentre la funzione che calcola n come nell'enunciato supera abbondantemente questo limite, e pertanto il teorema non è dimostrabile nella teoria.

Teorema di Kruskal

Il teorema di Kruskal gioca un ruolo importante nell'algebra dei quasi ordini, un argomento che ha mostrato una notevole rilevanza nella dimostrazione di terminazione di algoritmi complessi.

Questo fatto rende il teorema di Kruskal significativo in ambito matematico, e risponde alla questione se esistano proprietà naturali indimostrabili all'interno dell'aritmetica di Peano.

Quasi ordini

Definizione 24.3 (Buon quasi ordine)

Un *quasi ordine* è una struttura $\langle \mathcal{O}; \leq \rangle$ tale che $\leq$ è una relazione binaria su $\mathcal{O}$ che sia riflessiva e transitiva.

Un *buon quasi ordine* è un quasi ordine tale che

- ogni catena discendente propria è finita: una *catena discendente propria* è una sequenza $\{e_i\}_i$ di elementi di $\mathcal{O}$ tale che $e_i < e_j$ se $j < i$;
- ogni anticatena è finita: una *anticatena* è un sottoinsieme $A \subseteq \mathcal{O}$ tale che, se $a, b \in A$ e $a \neq b$, allora $a \not\leq b$ e $b \not\leq a$.

Teorema di Kruskal

L'importanza dei buoni quasi ordini deriva da una generalizzazione del teorema di Kruskal, il quale afferma

Teorema 24.4 (Kruskal)

L'insieme degli alberi finiti con la relazione di immersione forma un buon quasi ordine.

La relazione di immersione è definita come: $T \leq S$ se e solo se esiste una funzione iniettiva f dai nodi di T ai nodi di S che preservi i percorsi, ovvero se c'è un ramo da a a b in T, allora esiste un ramo da $f(a)$ a $f(b)$ in S.

Minori di grafi

Definizione 24.5 (Minore di un grafo)

Siano $\mathscr{G}$ e $\mathscr{H}$ due grafi finiti non diretti. Allora $\mathscr{H}$ è un *minore* di $\mathscr{G}$ se e solo se esiste una relazione di equivalenza $\sim$ tra i nodi di $\mathscr{G}$ e una funzione iniettiva dai nodi di $\mathscr{H}$ ai nodi di $\mathscr{G}$ tali che

- se $a \sim b$ allora esiste un percorso da a a b;
- se esiste l'arco (a, b) in $\mathscr{H}$, allora esistono due nodi c e d in $\mathscr{G}$ tali che $f(a) \sim c$, $f(b) \sim d$ e (c, d) è un arco in $\mathscr{G}$.

Minori di grafi

L'idea di minore, al di là della definizione tecnica, significa che possiamo partizionare i nodi di $\mathscr{G}$ in sottoinsiemi connessi, e possiamo considerare il grafo $\mathscr{G}/\sim$ i cui nodi siano questi sottoinsiemi e i cui archi siano gli archi tra nodi di sottoinsiemi differenti. Allora $\mathscr{H} \leq \mathscr{G}$ se $\mathscr{H}$ è un sottografo di $\mathscr{G}/\sim$.

Un applicazione naturale della nozione di minore si riscontra nelle reti di telecomunicazione.
Per trasportare un'informazione da un nodo ad un altro, serve calcolare un percorso che connetta i due nodi. In una rete enorme, è verosimile che nessun nodo contenga la mappa di tutta la rete, ma soltanto una sottoparte che 'riassuma' un pezzo di tutta la rete. La relazione $\sim$ modella il fatto che una sottorete viene riassunta in un solo nodo, mentre il grafo $\mathscr{G}/\sim$ modella la rete complessiva in cui le sottoreti locali siano considerate come nodi. La relazione di minore modella che il fatto che la mappa della sottoparte sia coerente con la mappa globale.

Minori di grafi

Teorema 24.6 (Minori di grafi)

L'insieme dei grafi finiti non diretti con la relazione di minore tra grafi forma un buon quasi ordine.

Questo teorema, la cui dimostrazione costituisce uno dei grandi risultati della matematica del XX secolo, è evidentemente non dimostrabile nell'aritmetica di Peano in quanto permette di derivare direttamente il teorema di Kruskal.

Uno dei sottoprodotti importanti della dimostrazione è un algoritmo polinomiale per calcolare se un grafo è un minore di un altro.

Riferimenti

Il teorema di incompletezza debole è folklore.

Il lavoro originale *J.B. Kruskal*, Well-quasi-ordering, the tree theorem, and Vazsonyi's conjecture, Transactions of the American Mathematical Society 95(2), pp. 210–225, American Mathematical Society, (1960), è una eccellente introduzione al teorema di Kruskal e alle sue motivazioni.

Sebbene vi siano numerosi testi che diano una panoramica generale della combinatorica, il mio preferito è *M. Bóna*, A Walk Through Combinatorics, 2ª edizione, World Scientific, (2006).