

Linguaggi di Programmazione

Daniele De Micheli

2019

Indice

I	Introduzione alla logica	1
1	Logica e Ragionamento	1
1.1	Processo di dimostrazione	3
1.2	Regole di inferenza e calcoli logici	3
2	Programmazione Logica	4
2.1	Stile dichiarativo della programmazione logica	4
2.2	PROLOG	5
2.2.1	Forma Normale Congiuntiva	5
2.2.2	Struttura del Prolog	6
2.2.3	Sintassi del Prolog	7

Parte I

Introduzione alla logica

1 Logica e Ragionamento

Per poter iniziare a parlare di *linguaggi logici*, dobbiamo prima acquisire cosa è un *linguaggio*. Dobbiamo quindi capire come un **ragionamento** può essere **formalizzato** in un numero di **passi** (connessi da **regole**) a partire da **premesse** per raggiungere una **conclusione**.

Questo processo è quello che siamo abituati a riscontrare nella soluzione di *teoremi* tramite **dimostrazioni**.

Un esempio di applicazione di questo processo possiamo vederlo qui di seguito:

Teorema del triangolo isoscele. *Dato un triangolo isoscele, ovvero con due lati $AB = BC$, si dimostra che gli angoli $\angle A$ e $\angle C$ sono uguali.*

Conoscenze pregresse

1. Se due triangoli sono uguali, i due triangoli hanno lati e angoli uguali.
2. Se due triangoli hanno due lati e l'angolo sotteso uguali, allora i due triangoli sono uguali.
3. BH bisettrice di $\angle B$ cioè $\angle ABH = \angle HBC$.

Dimostrazione

- $AB = BC$ per ipotesi;
- $\angle ABH = \angle HBC$ per (3);
- Il triangolo HBC è uguale al triangolo ABH per (2);
- $\angle A$ e $\angle C$ per (1);

Quindi abbiamo trasformato (2) in "**Se** $AB = BC$ e $BH = BH$ e $\angle ABH = \angle HBC$, **allora** il triangolo ABH è uguale al triangolo HBC " e abbiamo trasformato (1) in "**Se** triangolo ABH è uguale al triangolo HBC , **allora** $AB = BC$ e $BH = BH$ e $AH = HC$ e $\angle ABH = \angle HBC$ e $\angle AHB = \angle CHB$ e $\angle A = \angle C$ ".

L'obiettivo diventa a questo punto formalizzare e razionalizzare il processo che permette di affermare

$$AB = BC \vdash \angle A = \angle C$$

dove \vdash indica il simbolo di *derivazione logica*, che comunemente significa "consegue", "allora", ecc.

Formalizzazione

Abbiamo assunto che:

- $\mathbf{P} = \{AB = BC, \angle ABH = \angle HBC, BH = HB\}$.

Avevamo inoltre delle conoscenze pregresse (vedi *conoscenze pregresse* sopra riportate). Abbiamo quindi costruito una catena di **formule**: •

Le parti evidenziate in rosso nella tabella 1 sono le *regole di inferenza*.

Tabella 1: Triangolo Isoscele.

Formule	Origine
P1: $AB = BC$	da P 10.1
P2: $\angle ABH = \angle HBC$	da P
P3: $BH = HB$	da P
P4: $AB = BC \wedge BH = HB \wedge \angle ABH = \angle HBC$	da P1, P2, P3 e introduzione della congiunzione
P5: $\triangle ABH = \triangle HBC$	da P4 , regola ₁ e Modus Ponens
P6: $AB = BC \wedge BH = HB \wedge AH = HC$ $\wedge \angle ABH = \angle HBC \wedge \angle AHB = \angle CHB \wedge$ $\angle A = \angle C$	da P5 , regola ₂ e Modus Ponens
P7: $\angle A = \angle C$	da P6 e l'eliminazione della congiunzione (il simbolo \wedge)

1.1 Processo di dimostrazione

Una "prova" D , dove **S** è l'insieme delle "affermazioni note" e **F** la frase (es. la formula) che vogliamo provare.

$$D \equiv \mathbf{S} \vdash F$$

(che si legge: **F** è una conseguenza di **S**) è una sequenza di passi

$$D = \langle P_1, P_2, \dots, P_n \rangle$$

dove

$$P_n = F$$

$$P_i \in S \cup \{P_j \mid j < i\}$$

o P_i può essere ottenuto da P_{i_1}, \dots, P_{i_m} (con $i_1 < i, \dots, i_m < i$) mediante l'applicazione di una regola di inferenza.

1.2 Regole di inferenza e calcoli logici

Un insieme di regole di inferenza costituisce la base di un calcolo logico. Diversi insieme di regole danno vita a diversi calcoli logici. Lo scopo di un calcolo logico è quello di manipolare delle formule logiche in modo completamente **sintattico**

al fine di stabilire una connessione tra un insieme di formule di *partenza* (di solito un insieme di formule dette *assiomi*) ed un insieme di *conclusioni*.

2 Programmazione Logica

La programmazione logica nasce all'inizio degli anni settanta da studi sulla deduzione automatica: il **Prolog** costituisce uno dei suoi risultati principali. Essa non è soltanto rappresentata dal Prolog; costituisce infatti un settore molto ricco che cerca di utilizzare la logica matematica come base dei linguaggi di programmazione.

Gli obiettivi del linguaggio di programmazione logica sono:

- semplicità del formalismo;
- linguaggio ad alto livello;
- semantica chiara;

Questo tipo di linguaggio si focalizza sulle solide basi della *logica matematica*. Con l'avvento dell'informatica difatti si è sempre più utilizzata la logica matematica per dimostrare teoremi tramite i calcolatori (che permettono di ottenere risultati in minor tempo e con meno errori). Tra le procedure utilizzate si ricordano la *procedura di Davis e Putnam* e il *principio di risoluzione*.

Per rendere bene l'idea, la programmazione logica viene utilizzata per verificare la correttezza di altri software, per rappresentare la conoscenza di Intelligenza Artificiale o ancora per il formalismo nei database (come Datalog).

2.1 Stile dichiarativo della programmazione logica

Lo stile della programmazione logica ha delle precise caratteristiche:

- Un programma è un *insieme di formule*.
- Possiede un grande potere espressivo.
- Il processo di risoluzione prevede la costruzione di una dimostrazione logica di un'affermazione (**goal**).
- Possiede una **base formale**:
 - Calcolo dei predicati del primo ordine (vedi sez. ????) ma con limitazione nel tipo di formule (**clausole di Horn** (1))

- Utilizzo di particolari tecniche per la dimostrazione di teoremi (meccanismo di **Risoluzione**)

2.2 PROLOG

Il Prolog (acronimo di **PRO**gramming in **LOGic**) fu ideato e realizzato nel 1973 da Robert Kowalski (aspetto teorico) e Marten Van Emdem (dimostrazione sperimentale). Esso si basa su una restrizione della *logica del primo ordine*. Come caratteristica base dei linguaggi logici, anche Prolog utilizza uno stile dichiarativo di programmazione. La sua primaria funzione è quella di determinare se una certa affermazione è vera oppure no e, se è vera, quali vincoli sui valori attribuiti alle variabili hanno generato la risposta.

Formule ben formate Le formule ben formate (*fbf*, o **well-formed formula**, *wff*) di un linguaggio logico del primo ordine può essere riscritta in **forma normale a clausole**.

Vi sono due forme normali a clausole:

- **Forma normale congiunta** (conjunctive normal form - CNF): la formula è una **congiunzione** di **disgiunzioni** di predicati o di negazioni di predicati (letterali *positivi* o letterali *negativi*).

$$\bigwedge_i \left(\bigvee_j L_{ji} \right) \quad (1)$$

- **Forma normale disgiunta** (disjunctive normal form - DNF): la formula è una **disgiunzione** di **congiunzioni** di predicati o di negazione di predicati (letterali *positivi* o letterali *negativi*).

$$\bigvee_j \left(\bigwedge_i L_{ji} \right) \quad (2)$$

dove

$$L_{ij} \equiv P_{ij}(x, y, \dots, z) \circ L_{ij} \equiv \neg Q_{ij}(x, y, \dots, z)$$

2.2.1 Forma Normale Congiuntiva

Consideriamo una *wff* in CNF (1). Per esempio:

$$\underbrace{(p(x) \vee q(x, y) \vee \neg t(z))}_{\text{clausola1}} \wedge \underbrace{(p(w) \vee \neg s(u) \vee \neg r(v))}_{\text{clausola2}} \quad (3)$$

se scartiamo il simbolo di **congiunzione** (3), rimaniamo con solo le clausole disgiuntive

1. $(p(x) \vee q(x, y) \vee \neg t(z))$
2. $(p(w) \vee \neg s(u) \vee \neg r(v))$

Le clausole così ottenute sono anche riscrivibili come

1. $t(z) \Rightarrow p(x) \vee q(x, y)$
2. $s(u) \wedge r(v) \Rightarrow p(w)$

ovvero un insieme di formule in CNF è riscrivibile come un insieme (congiunzione) di implicazioni.

Clausole di Horn 1. *Le clausole che hanno al più un solo letterale positivo (con o senza letterali negativi) prendono il nome di **Clausole di Horn**.*

Detto questo, abbiamo che:

- Non tutte le fbf possono essere trasformate in un insieme di clausole di Horn.
- *I programmi Prolog sono collezioni di clausole di Horn*

2.2.2 Struttura del Prolog

Il linguaggio Prolog:

- Non contiene (quasi) istruzioni
- Contiene solo fatti e regole (clausole di Horn):
 - **Fatti**: asserzioni vere nel contesto che stiamo descrivendo.
 - **Regole**: ci danno gli strumenti per dedurre nuovi fatti da quelli esistenti.
- Un programma Prolog ci dà informazioni su un sistema ed è normalmente chiamato **base di conoscenza** (knowledge base)
- Un programma Prolog non si "esegue" ma si "interroga" (**queried**): al programma si chiede se i fatti sono veri; esso risponderà con un **True** o **False** alla domanda.

Tabella 2: Sintassi Prolog

$a.$	FATTO / ASERZIONE
$c : - b_1, b_2, b_n.$	REGOLA
$:- q_1, q_2, \dots, q_m$	GOAL
$? - q_1, q_2, \dots, q_m$	QUERY

2.2.3 Sintassi del Prolog




Un programma Prolog è costituito da un insieme di clausole della forma

In cui a, b_i e q_i sono **termini** (compositi). Da notare che in molte implementazioni il prompt Prolog è un operatore che chiede al sistema di valutare il **goal**, in questo caso una congiunzione di termini.

Termini Le espressioni nel Prolog sono chiamate **termini**. Esistono diversi tipi di termini:

- ATOMI :
 - Una sequenza di caratteri alfanumerici, che inizia con un carattere minuscolo (può contenere il carattere " " underscore)
 - Qualsiasi cosa racchiusa tra apici singoli (' ')
 - Un numero
 - Una stringa (SWI Prolog)
- VARIABILI:
 - Una variabile (logica) è una sequenza alfanumerica che inizia con un carattere **MAIUSCOLO** o con il carattere _ (underscore)
 - Le variabili (notare il plurale) composte solo dal simbolo _ sono chiamate **indifferenza** (*don't care o any*) o **anonime**
 - Le variabili vengono **istanziate** (legate a un valore) con il procedere del programma (nella dimostrazione del teorema)
- Una composizione di termini \Rightarrow TERMINE COMPOSTO (simbolo di **fun-tore** più uno o due argomenti):

- + Un **funtore** (simbolo di funzione o di predicato definito come atomo)
- + Una sequenza di termini racchiusi tra parentesi tonde e separati da virgole. Questi sono chiamati argomenti del funtore

   ***Non ci deve mai essere uno spazio tra il funtore e la parentesi di sinistra;*** questo per via di caratteristiche molto sofisticate del sistema di parsing di Prolog (cfr., operatori)