Linguaggi di Programmazione

Daniele De Micheli

2019

Indice

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:

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: $AB = BC \bigwedge BH = BH \bigwedge AH = HC \bigwedge \angle ABH = \angle HBC \bigwedge \angle AHB = \angle CHB \bigwedge \angle A = \angle C$	da **P5**, $regola_2$ e Modus Ponens
P7: $\angle A = \angle C$	da **P6** e l'eliminazione della congiunzione(il simbolo $\bigwedge$)

- **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*.

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

$$D =< P_1, P_2, ..., P_n >$$

dove

$$P_n = F$$

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

o P_i può essere ottenuto da $P_{i1}, ..., P_{im}$ (con $i1 < i, ..., im < i$)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 sui 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 **LOG**ic) 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*).

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

- 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*).

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

dove

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

2.2.1 Forma Normale Congiuntiva

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

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

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 son un True o False alla domanda.

2.2.3 Sintassi del Prolog

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

Tabella 2: Sintassi Prolog

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

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 ⇒ TERMINE COMPOSTO (simbolo di *funtore* 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)

⚠⚠ ⚠⚠

Prolog: fatti e predicati Un **fatto** (**predicato**) consiste in:

- Un nome di predicato, ad esempio *fattoriale*, *genitore*, *uomo* o *animale*; deve iniziare con una lettera minuscola.
- Zero o più argomenti come *maria*, *42* o *cane*. Da tenere presente che i fatti (e le regole e le domande) **devono** essere terminati da un punto (".").

Le regole In Prolog si usano le **regole** quando si vuole esprimere che un certo fatto dipende da un insieme di altri fatti. Per esprimere in linguaggio naturale questa dipendenza usiamo la parola "se" ("if" in inglese).

Ad esempio:

- *Uso l'ombrello **se** piove*;
- *Luca mangia la pizza **se** non contiene glutine*;

Le regole sono anche usate per esprimere definizioni. Ad esempio:

X è un pesce se:

- X è un animale
- X ha le squame

Una regola è costituita da una **testa** e da un **corpo**. Sono inoltre caratterizzati come segue:

1. Testa e corpo sono collegati dalloperatore ": −".
2. La testa di una regola corrisponde al **conseguente** di un'implicazione logica.
3. Il corpo di una regola corrisponde all'**antecendente** di un'implicazione logica.
4. Le regole Prolog corrispondono alle **clausole di Horn** , ovvero hanno un solo termine (predicato) come conseguente.
    - L'operatore Prolog ': −' esprime il "**se**" (*implicazione*).
    - L'operatore Prolog ',' equivale a "**e**" (***and***, o *congiunzione*).

Un esempio di regola Prolog può essere:

"un pesce è un animale che ha le squame" diventa:

$pesce(X) :- animale(X), ha\_le\_squame(X).$

Una relazione può essere però definita per necessità da due regole (o clausole) aventi lo stesso predicato come conclusione. Per esempio:

$genitore(X,Y) :- padre(x,Y).$
$genitore(X,Y) :- madre(x,Y).$

Le regole (ed i fatti) sono implicitamente connesse dall'operatore logico di congiunzione ("and"); se non si sono commessi errori logici, **entrambe** le implicazioni soprastanti sono da ritenersi **vere**.

Una relazione può essere definita anche *ricorsivamente*. In questo caso la definizione richiede almeno due proposizioni: una è quella ricorsiva che corrisponde al caso generale, l'altra esprime il caso base più semplice.

$antenato(X,Y) :- genitore(X,Y).$
$antenato(X,Y) :- genitore(Z,Y), antenato(X,Z).$

Operatori Logici : gli operatori logici "and" e "or" possono essere utilizzati nelle regole tramite una specifica sintassi. L'operatore "**AND**" viene inserito in una regola con il carattere *virgola* ','; L'operatore "**OR**" viene inserito in una regola con il carattere *punto e virgola* ';'.

RICAPITOLANDO Bisogna ricordare che:

1. ogni fatto o regola DEVE terminare con un punto '.'
2. ogni variabile DEVE iniziare con una MAIUSCOLA
3. i commenti si inseriscono dopo un "%" (commento in linea) o tra '/*' e '*/' (come in altri linguaggi).

2.3 L'interprete Prolog: interrogazioni

Una volta che le regole e i fatti sono stati "caricati" nell'interprete, eseguire un programma Prolog equivale a *interrogare* l'**interprete**. Una volta fatto partire, l'interprete Prolog ci presenta un prompt così:

?–

Come si può intuire anche dalla sintassi del prompt, interrogare l'interprete non è altro che porre una *domanda* a cui generlamente il Prolog risponde "**true**" o "**false**". Le interrogazioni possono contenere variabili interpretate come ***variabili esistenziali***. Queste sono istanziate quando il Prolog prova a rispondere alla domanda fattagli. Tutte le variabili istanziate vengono mostrate nella risposta insieme a "true".

2.3.1 L'interprete prolog: consult

Nel Prolog la base di conoscenza è *nascosta* ed è accessibile solo tramite opportuni comandi o tramite ambiente di programmazione. Ovviamente è necessario poter *inizializzare* o *caricare* un insieme di fatti e regole nell'ambiente Prolog. Il comando principale che assolve questa funzione è **consult**:

- Il comando consult appare come un predicato da valutare (un goal) e prende almeno un termine che denota un file come argomento.
- Il file deve contenere un insieme di fatti e regole.

2.4 Unificazione

L'operazione di istanziazione di variabili durante la "prova" di un predicato è il risultato di una procedura particolare, detta unificazione. Dati due termini, la procedura di unificazione crea un insieme di *sostituzioni* delle variabili, il quale permette di rendere "**uguali**" i due termini.

Tradizionalmente la procedura di unificazione costruisce un insieme di sostituzioni chiamato "most general unifier" ed indicato con *Mgu*. Una sostituzione è indicata come una sequenza (ordinata) di coppie variabile/valore.

Esempi:

$Mgu(42,42) \Rightarrow \{\}$

$Mgu(42,X) \Rightarrow \{X/42\}$

$Mgu(X,42) \Rightarrow \{X/42\}$

$Mgu(foo(bar,42),foo(bar,X)) \Rightarrow \{X/42\}$

$Mgu(foo(Y,42),foo(bar,X)) \Rightarrow \{X/42\}$

$Mgu(foo(X),foo(bar(Y))) \Rightarrow \{X/bar(Y),Y/\_G001\}$

Notare l'ultimo esempio con **ridenominazione** di variabili (_ G001 è il nome assegnato alla variabile che contiene il risultato. Esso è generato dal programma Prolog). Il "*most general unifier*" non è nient'altro che il risultato finale della procedura di valutazione - ovvero di prova - del Prolog.

Il modo più semplice per vedere come la procedura di unificazione funziona è di usare l'operatore Prolog = (detto, per l'appunto, operatore di unificazione).

Esempi:

?– **42 = 42.**

Yes

?– **42 = X.**

$X = 42$

Yes

?– **foo(Y, 42)=foo(bar, X).**

$Y = bar$

$X = 42$

Yes

2.5 Liste in Prolog

Si definisce una *lista* in Prolog racchiudendo gli elementi (termini e/o variabili logiche) della lista tra parentesi quadre '[' e ']' e separandoli da virgole. Gli elementi di una lista in Prolog possono essere termini qualsiasi o liste. Per indicare una lista vuota si usa la notazione '[ ]'.

Ogni lista non vuota può essere divisa in due parti, una *testa* ed *coda* :

- La *testa* è il primo elemento della lista.
- La *coda* rappresenta tutto il resto della lista ed è **SEMPRE** una lista.

2.5.1 L'operatore |

Prolog possiede uno speciale operatore usato per distinguere tra l'inizio e la coda di una lista: l'operatore "|".

Esempi dell'operatore | :

? – **[X | Ys] = [mia, vincente, jules, yolanda].**

$X = mia$

$Ys = [vincent, jules, yolanda]$

Yes

? – **[X, Y | Zs] = [the, answer, is, 42].**

$X = the$

$Y = answer$

$Zs = [is, 42]$

Yes

? – **[X, 42 | _] = [41, 42,43, foo(bar)].**

$X = 41$

Yes