DIPARTIMENTO DI INFORMATICA SISTEMISTICA E COMUNICAZIONE

Linguaggi di Programmazione 2022-2023
Prolog e programmazione logica I

Marco Antoniotti
Gabriella Pasi
Rafael Peñaloza

Prolog

- Questa parte del corso è dedicata al linguaggio Prolog ed al paradigma di **programmazione logica**

- Il libro di testo è *The Art of Prolog* di Sterling e Shapiro, MIT Press (oppure I. Bratko, *Prolog Programming for Artificial Intelligence*)

- Sistema consigliato: SWI-Prolog

Programmazione logica

- La programmazione logica nasce all'inizio degli anni settanta da studi sulla deduzione automatica: il Prolog costituisce uno dei sui risultati principali.
- La programmazione logica **NON** è soltanto rappresentata dal Prolog: essa costituisce infatti un settore molto ricco che cerca di utilizzare la logica matematica come base dei linguaggi di programmazione. Prolog è l'esempio più noto.

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

Programmazione logica e la logica matematica

- Logica matematica: formalizzazione del ragionamento

- Con l'avvento dell'informatica:
  - Logica Matematica $\Rightarrow$ Dimostrazione automatica di teoremi (procedura di Davis e Putnam e principio di risoluzione)
  - Interpretazione procedurale di formule (Kowalski)
    - Logica come linguaggio di programmazione

- Utilizzata in varie applicazioni: dalle prove di correttezza dei programmi alle specifiche, da linguaggio per la rappresentazione della conoscenza in Intelligenza Artificiale a formalismo per DB (esempio, `datalog`).

Stile dichiarativo della programmazione logica

- Programma come insieme di formule
- Grande potere espressivo
- Computazione come costruzione di una dimostrazione di una affermazione (goal)

- Base formale
  - Calcolo dei predicati del primo ordine ma con limitazione nel tipo di formule (clausole di Horn)
  - Utilizzo di particolari tecniche per la dimostrazione di teoremi (meccanismo di Risoluzione)

Prolog

- Acronimo per **PRO**gramming in **LOG**ic. Nato nel 1973.
- Basato su una restrizione della logica del primo ordine
- Stile dichiarativo
- Prolog è usato per determinare se una certa affermazione è vera o no e, se è vera, quali vincoli sui valori attribuibili alle variabili hanno generato la risposta
  - Nuovi sviluppi
    - Constraint Logic Programming (CLP)

Formule ben formate e forma normale "a clausole"

- Ogni formula ben formata (*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 e **letterali** negativi)
  - **Forma normale disgiunta** (disjunctive normal form - DNF)
    - La formula è una ***disgiunzione*** di ***congiunzioni*** di predicati o di negazioni di predicati (**letterali** positivi e **letterali** negativi)

Formule ben formate e forma normale "a clausole"

- **Forma normale congiunta** (conjunctive normal form - CNF)

$$\bigwedge_i \left(\bigvee_j L_{ij} \right)$$

- **Forma normale disgiunta** (disjunctive normal form - DNF)

$$\bigvee_j \left(\bigwedge_i L_{ij} \right)$$

dove

$$L_{ij} \equiv P_{ij}(x, y, \cdots, z) \quad \text{o} \quad L_{ij} \equiv \neg Q_{ij}(x, y, \cdots, z)$$

Forma Normale Congiuntiva

- Consideriamo una wff in CNF $\underbrace{\bigwedge_i \left(\bigvee_j L_{ij} \right)}_{\text{clausole}}$
- Esempi

$$\underbrace{(p(x) \vee q(x,y) \vee \neg t(z))}_{\text{clausola 1}} \wedge \underbrace{(p(w) \vee \neg s(u) \vee \neg r(v))}_{\text{clausola 2}}$$

$$\underbrace{(\neg t(z))}_{\text{clausola 1}} \wedge \underbrace{(p(w) \vee \neg s(u))}_{\text{clausola 2}} \wedge \underbrace{(p(x) \vee s(x) \vee q(y))}_{\text{clausola 3}}$$

se scartiamo il simbolo di congiunzione, rimaniamo con solo le clausole disgiuntive (primo esempio):

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

Forma Normale Congiuntiva

- Le clausole relative al primo esempio sono anche riscrivibili come

$$t(z) \Rightarrow p(x) \vee q(x,y)$$

$$s(u) \wedge r(v) \Rightarrow p(w)$$

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

- Le clausole che hanno al più un solo letterale positivo (con o senza letterali negativi) si chiamano **clausole di Horn**
  - Non tutte le fbf possono essere trasformate in un insieme di clausole di Horn
  - **I programmi Prolog sono collezioni di clausole di Horn**
    - Grazie a questa restrizione, Kowalski, nel 1974, produsse una interpretazione procedurale di un insieme di clausole (di Horn) che è alla base della semantica di tutti i sistemi Prolog (Kowalski, R., *Predicate Logic as Programming Language*, in Proceedings IFIP Congress, Stockholm, North Holland Publishing Co., 1974, pp. 569-574)

Prolog – Linguaggio dichiarativo

- Non contiene istruzioni (quasi)
- 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**)
  - Ci si chiede: questi fatti sono veri?
  - Il Programma risponderà solo **SI** o **NO** alla domanda

Programmi Prolog

- **Sintassi**: un programma Prolog è costituito da un insieme di clausole della forma

```
a.                          % FATTO o ASSERZIONE
c :- b1, b2, …, bn.         % REGOLA

:- q1, q2, …, qm.           % GOAL
?- q1, q2, …, qm.           % QUERY
```

- In cui **a**, $\mathbf{b_i}$, **c**, e $\mathbf{q_i}$ sono **termini** (composti)
- Notare che in molte implementazioni il prompt Prolog è anche un operatore che chiede al sistema di valutare il **goal**, in questo caso una congiunzione di termini

Termini

- Le espressioni del Prolog sono chiamate **TERMINI**. Esistono diversi tipi di termini:
  - Atomi
    - Numeri
    - …
  - Variabili
  - Una composizione di termini ⇒ termine composto (simbolo di *funtore* + uno o più argomenti)

Atomi

- Un **atomo** è:
  - Una sequenza di caratteri alfanumerici, che inizia con un carattere **minuscolo** (può contenere il carattere "_" underscore)
  - Qualsiasi cosa racchiusa tra apici (' ')
  - Un numero
  - Una stringa (SWI Prolog)
  - …

Variabili (logiche)

- 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)

Termini composti

- Una composizione di termini consiste in:
  - 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**)

Esempi di termini

- Validi

```
foo             hello
Hello           sam
sam             hello_Sam
hello-sam       40+2
quarantaquattro-4
'hello'         'Hello'
'Hello Sam'     _
a1              '1a'
X               _hello
_234            hello(X)
f(a)            f(a, b, c)
f(hello, Sam)  f( a, b, c )
p(f(a), b)
hello(1, hello(x, X, hello(sam)))
t(a, t(b, t(c, t(d, []))))
```

- Non validi

```
hello Sam       Hello Sam
hello sam       1a
f(a, b          f(a,
f a, b)         f (a, b)
X(a, b)         1(a, b)
```

Prolog: i fatti o 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 notare che i fatti (e le regole e le domande) devono essere terminati da un punto "**`.`**" .

Fatti: esempi

- Esempi

```
libro(kowalski, prolog).
libro(yoshimoto, kitchen).
donna(yoshimoto).
uomo(kowalski).
animale(cane).
animale(trota).
ha_le_squame(trota).
la_risposta(42).
```

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)
  - *Uso l'ombrello **se** piove.*
  - *Gino compra il vino **se** è meno caro della birra.*

Le regole

- Le regole sono anche usate per esprimere definizioni:
  - X è un pesce **se**:
    - *X è un animale*
    - *X ha le squame*
- oppure
  - X è sorella di Y **se**:
    - *X è di sesso femminile*
    - *X e Y hanno gli stessi genitori*

Le regole

- In Prolog una regola consiste di una **testa** (head) e di un **corpo** (body)
  - Testa e corpo sono collegati dall'operatore `:-`
  - La testa di una regola corrisponde al **conseguente** di un'implicazione logica
  - Il corpo di una regola corrisponde all'**antecedente** di un'implicazione logica
  - 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*)

Regole

- Esempi
  - La frase "*un pesce è un animale che ha le squame*" in Prolog diventa:

```
pesce(X) :- animale(X), ha_le_squame(X).
```

  - La frase *"Gigi ama chiunque ami il vino"* in Prolog diventa:

```
ama(gigi, X) :- ama(X, vino).
```

  - La frase *"Gigi ama le donne a cui piace il vino"*, in Prolog sarà:

```
ama(gigi, X) :- donna(X), ama(X, vino).
```

Prolog e relazioni definite da più regole

- Una relazione (ad esempio **`genitore`**) può essere definita da più regole (ovvero da più clausole) aventi lo stesso predicato come conclusione.

- 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 – per l'appunto – **logici**, **entrambe** le implicazioni qua sopra sono da ritenersi **vere**

Regole – La ricorsione

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

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

Operatori logici

- L'operatore logico **AND** inserito in una regola viene espresso mediante la virgola '`,`'
- L'operatore logico **OR** inserito in una regola viene invece espresso mediante il punto e virgola '`;`'

Sintassi

- Ricordatevi che
  - ogni fatto o regola o funzione DEVE terminare con un punto '.'
  - ogni variabile DEVE iniziare con una maiuscola
  - I commenti si inseriscono dopo un «%» (commento di linea) o tra «/*» e «*/» (come in C/C++ e Java)

```
%%% Questa è una linea di commento
f(a,b). % Solo questa parte di riga è un commento.

%%% Ecco un commento
%%% su più righe.
%%% Anche questo è un commento.

/* Questo commento può essere
 * scritto su più righe
 */
```

L'interprete Prolog: interrogazioni (queries o goals)

- Una volta che le regole ed i fatti sono scritte e caricate nell'interprete (vedremo come), eseguire un programma Prolog significa **interrogare** l'**interprete**
- Una volta fatto partire, di solito, l'interprete Prolog ci presenta il prompt

```
?-
```

- Un esempio di query diventa quindi

```
?- libro(kowalski, prolog).
```

- Dati i fatti e le regole che abbiamo visto precedentemente, il Prolog risponde «Yes» o «true»
  - *Interrogare il programma equivale a richiedere la dimostrazione di un teorema*

Le variabili delle interrogazioni

- Le interrogazioni possono contenere variabili interpretate come variabili *esistenziali*
- Le variabili sono *istanziate* quando il Prolog prova a rispondere alla domanda
- Tutte le variabili istanziate vengono mostrate nella risposta.

- Esempi

```
?- libro(kowalski, LINGUAGGIO).
Yes
   LINGUAGGIO = prolog

?- libro(AUTORE, prolog).
Yes
   AUTORE = kowalski

?- libro(AUTORE, LINGUAGGIO).
Yes
   AUTORE = kowalski
   LINGUAGGIO = prolog
```

Unificazione: introduzione

- 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

- Questo insieme di sostituzioni (brevemente: sostituzione o assegnamento) permette di rendere «**uguali**» i due termini

Unificazione: introduzione

- 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`)	⇒ {}
Mgu(`42, X`)	⇒ {`X/42`}
Mgu(`X, 42`)	⇒ {`X/42`}
Mgu(`foo(bar, 42), foo(bar, X)`)	⇒ {`X/42`}
Mgu(`foo(Y, 42), foo(bar, X)`)	⇒ {`Y/bar, X/42`}
Mgu(`foo(bar(42), baz), foo(X, Y)`)	⇒ {`X/bar(42), Y/baz`}
Mgu(`foo(X), foo(bar(Y))`)	⇒ {`X/bar(Y), Y/_G001`)}

- Notare l'ultimo esempio con **ridenominazione** di variabili

Unificazione: introduzione

- 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(bar, 42) = foo(bar, X).
   X = 42
   Yes
```

Unificazione: introduzione

- **Esempi**

```
?- foo(Y, 42) = foo(bar, X).
   Y = bar
   X = 42
   Yes

?- foo(bar(42), baz) = foo(X, Y).
   X = bar(42)
   Y = baz
   Yes

?- foo(X) = foo(bar(Y)).
   X = bar(Y)
   Y = _G001
   Yes

?- foo(42, bar(X), trillian) = foo(Y, bar(Y), X).
   No
```

Diverse rappresentazioni di dati ed interrogazioni

- Consideriamo il seguente esempio: vogliamo descrivere un insieme di fatti riguardanti i corsi offerti dal dipartimento (esempio dal capitolo 2 di “Art of Prolog”)

- **Possibilità 1**
  - Tutte le informazioni sono concentrate in una relazione con 6 campi

```
corso(linguaggi, lunedì, '9:30', 'U4', 3, antoniotti).
corso(biologia_computazionale, lunedì, '14:30', 'U14', t023, antoniotti).
```

  - A partire da questa definizione possiamo poi costruire altri predicati

```
aula(Corso, Edificio, Aula) :-
            corso(Corso, _, _, Edificio, Aula, _).
docente(Corso, Docente) :-
            corso(Corso, _, _, _, _, Docente).
```

Diverse rappresentazioni di dati ed interrogazioni (cont…)

- **Possibilità 2**
  - Tutte le informazioni sono concentrate in una relazione con 4 campi; le informazioni sono concentrate in termini funzionali che rappresentano informazioni raggruppate logicamente

```
corso(linguaggi, orario(lunedì, '9:30'), aula('U4', 3), antoniotti).
corso(biologia_computazionale,
      orario(lunedì, '14:30'),
      aula('U4', 3),
      antoniotti).
```

  - A partire da questa definizione possiamo poi costruire altri predicati

```
aula(Corso, Edificio, Aula) :- corso(Corso, _, aula(Edificio, Aula), _).
docente(Corso, Docente) :- corso(Corso, _, _, Docente).


%%%% oppure…

aula(Corso, Luogo) :- corso(Corso, _, Luogo, _).
```

Diverse rappresentazioni di dati ed interrogazioni (cont…)

- **Possibilità 3**
  - I predicati che abbiamo definito dalle relazioni con 6 o 4 campi possono essere ricodificate con predicati **binari** (cfr., le **triple** XML usate negli strumenti e nella ricerca sulla «semantic web»)

```
giorno(linguaggi, martedì).
orario(linguaggi, '9:30').
edificio(linguaggi, 'U4').
aula(linguaggi, 3).
docente(linguaggi, antoniotti).
```

  - La relazioni a 6 o 4 argomenti possono essere ricostruite in a partire da queste relazioni binarie
    - La costruzione di schemi RDF/XML (e, a volte, SQL) corrisponde a questa operazione di ri-rappresentazione con **triple**

NB, la rappresentazione qui sopra è insufficiente per rappresentare completamente un orario; perché?

Le 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
- `[]` indica la lista vuota.

Le liste: testa e coda

- Ogni lista non vuota può essere divisa in due parti: una testa ed una coda.
  - La testa è il primo elemento della lista
  - La coda rappresenta tutto il resto ed **è sempre una lista**

- **Esempi**

`[a, b, c]`	`a` è la testa e `[b, c]` la coda
`[a, b]`	`a` è la testa e `[b]` la coda
`[a]`	`a` è la testa e `[]` la coda
`[[a]]`	`[a]` è la testa e `[]` la coda
`[[a, b], c]`	`[a, b]` è la testa e `[c]` la coda
`[[a, b], [c], d]`	`[a, b]` è la testa e `[[c], d]` la coda

L'operatore |

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

- **Esempi** (notare la convenzione **`Ys`**, **`Zs`**)

```
?- [X | Ys] = [mia, vincent, 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
```

La lista vuota `[]`

- La lista vuota `[]` in prolog viene gestita come una lista speciale.

```
?- [X | Ys] = [].
No
```

L'interprete Prolog: `consult`

- 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

- **Esempi**

```
?- consult('guida-astrostoppista.pl').
Yes

?- consult('Projects/Lang/Prolog/Code/esempi-liste.pl').
Yes
```

L'interprete Prolog: `consult`

- Il predicato **`consult`** può essere usato anche per inserire fatti e regole direttamente alla console usando il termine speciale **`user`**

- **Esempi**

```
?- consult(user). % Notare il sotto-prompt.
|- foo(42).
|- friends(zaphod, trillian).
|- ^D
Yes

%%% A questo punto la base di dati Prolog contiene i due
%%% fatti inseriti manualmente.

?- friends(zaphod, W).
W = trillian
Yes
```

L'interprete Prolog: `reconsult`

- Il predicato **`reconsult`** deve invece essere usato quando si vuole ***ri***caricare un file (ovvero un data o knowledge base) nell'ambiente Prolog
- L'effetto è di prendere i predicati presenti nel file, rimuoverli completamente dal data base interno e di reinstallarli utilizzando le nuove definizioni (*)

- **Esempi**

```
?- reconsult('guida-astrostoppista.pl').
Yes

%%% A questo punto la base di dati Prolog contiene il
%%% nuovo contenuto del file.
```

(*) *La semantica di* **`consult`** *e* **`reconsult`** *è più complicata secondo lo standard ISO-Prolog*

Sommario

- Introduzione al Prolog
- **Termini**
  - Fatti
  - Regole
- **Unificazione**
- Rappresentare dati in Prolog
- **Liste**
- Minima introduzione all'ambiente Prolog