# Prolog

## Introduzione

- book: Prolog Programming for Artificial Intelligence
- SWI-Prolog
  - **listener**
    - richieste (query-goal) - interrogazioni al Knowledge Base
    - risposte (results) - di una richiesta
  - **database**
    - Knowledge Base
      - facts(arguments)
        - verifica se il fatto è vero o meno
        - risultato: true / false
        - i fatti con stesso predicato devono essere raggruppati insieme
      - rules(arguments)
        - verifica se la regola è vera o falsa
    - file con estensione `.pl`
- commenti `% commento`
- get current working directory: `working_directory(D,D).`
- set new working directory: `working_directory(D,"path").`
- print files in CWD: `ls.`
- switch db: `consult("DB.pl").`

## Unification

- le query in prolog includono il **pattern matching**
- query = richiesta = goal
- prolog pattern matching si chiama unification
- quando il DB contiene solo fatti, l'unificazione ha successo se:
  - il predicato usato come goal (query) e quello nel db corrispodono
  - i predicati hanno lo stesso numero di argomenti
  - tutti gli argomenti sono uguali
- Ex:

```
# DB family.pl
?- parent(tom,bob)       # Tom è padre di bob
parent/2            #  Altra forma

# Query
?- parent(bob,pat). # True -> è un fatto verificato

# Trovare tutti i genitori di bob
# Argomento che inizia con lettera maiuscola -> variabile
?- parent(X,bob).        # X = pam; X = tom.
```

```
# Mostrare tutti i figli di bob
?- parent(bob,X).       # X = ann; X = pat.

# Mostrare tutte le combinazioni genitore-figlio
?- parent(X,Y);     # X -> genitore, Y -> figlio

# Chi è il nonno di Jim?
# congiunzione (AND) -> ','
?- parent(X,jim),parent(Y,X).   # parent(Y,jim) AND parent(X,Y)

# Chi sono i nipoti di Tom?
?- parent(tom,Y),parent(Y,X).   # Da verificare

# Verificare se Ann e Pat hanno lo stesso genitore
?- parent(X,ann),parent(X,pat).
```

# Regole

- possiamo definire delle relazioni tramite regole

```
female(pam).
male(tom).
male(bob).
```

```
# Regola per definire la madre

# For all X and Y,
#   X is the mother of Y if
#       X is a parent of Y and X is female

mother(X,Y) :- parent(X,Y),female(X).

# For all X and Y,
#   X is a parent of Y then
#       X has a child
hasachild(X) :- parent(X,Y).
```

# Definizioni

- clause (clausole)
  - **fatti** - dichiarano cose che sono sempre vere
    - sono clause che hanno soltanto l'head o il body
  - **rules** - dichiarano cose che sono vere in base ad una condizione
    - formato da head e (non-empty) body
  - **questions** - per mezzo di esse l'utente può chiedere quali cose sono vere

```
# Everybody who has a child is happy
happy(X) :- hasChild(X).    # if X hasChild then X is happy

# For all X, if X has a child who has a sister then X has two children
hasTwoChildren(X) :- parent(X,Y),sister(Y,_).

# Define the grandchild relation using the parent one
grandchild(X,Y) :- parent(Y,Z),parent(Z,X).

# Define the aunt relation using the parent and sister ones
aunt(X,Y) :- parent(Z,Y),sister(X,Z).
```

# Ricorsione delle regole

- Es: mostrare gli antenati di X
- ricadiamo in un caso di ciclo infinito
- richiamo ricorsivo di una relazione logica fino ad arrivare ad un punto di arrivo

```
# Antenati a 2 livelli di gerarchia (nonno)
ancestor(X,Z) :-
    parent(X,Y1),
    parent(Y1,Z),

# Antenati a 3 livelli di gerarchia (bisnonno)
ancestor(X,Z) :-
    parent(X,Y1),
    parent(Y1,Y2),
    parent(Y2,Z).

# For all X and Z,
#   X is an ancestor of Z if
#       there is a Y such that
#           X is a parent of Y AND
#           Y is an ancestor of Z
```

# Data objects

- data objects
  - simple objects
    - constants
      - atoms (blocco che si sta usando)
      - numbers (integers, double)
    - variables
  - structures

# Atoms

- si possono costruire in tre modi:
  - stringhe con lettere, numeri, underscore (non all'inizio)
  - stringhe con caratteri speciali
    - `:-` è predefinita
  - stringhe che iniziano per maiuscolo tra virgolette

# Numeri

- Numeri reali e interi
- Notazione esponenziale: `7.15E-9`

# Variabili

- iniziano sempre con un carattere maiuscolo o un `_`
- Ex: `Anna`, `_anna`

```
# Stampa il nome del bambimo in caso positivo
has_a_child(X) :- parent(X,Y).

# Voglio sapere se ha un bambino senza sapere quale
has_a_child(X) :- parent(X,_).
```

- Ex:

```
# Creare dei predicati con un argomento che identifica il genere delle persone
di una famiglia
male(bob).
male(tom).
male(jim).
female(ann).
female(pam).
female(liz).
female(pat)

# Usare delle query che:
# 1. Confermano che una delle persone della famiglia sia maschio o femmina
?- male(bob).
# 2. Mostrano tutti i maschi della famiglia
?- male(X).
# 3. Mostrano tutte le femmine della famiglia
?- female(Y).
```