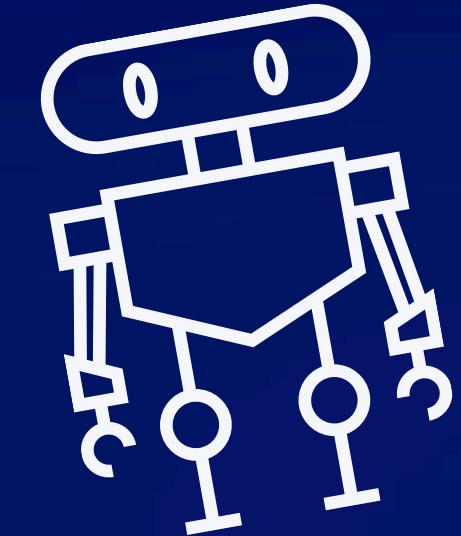
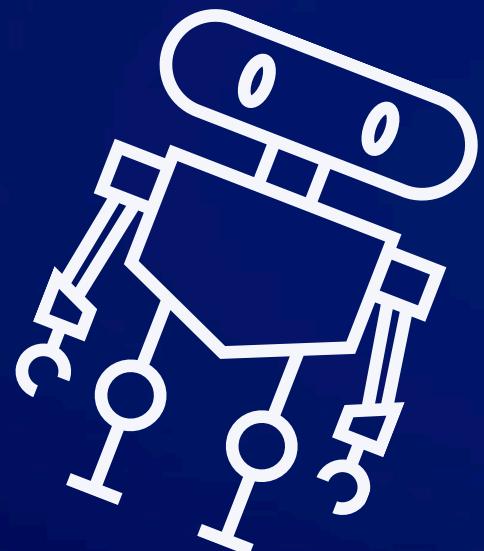




Introduzione a

PROLOG



Progetto Elena Azzi e Arianna Cella





Introduzione PROLOG

Programming in Logic

Prolog è un linguaggio di programmazione **logica dichiarativa**

Prolog è particolarmente adatto per problemi di:

Intelligenza
artificiale

Ingegneria del
software

Analisi del
linguaggio naturale

TIPI DI DATO



Costanti

Atomi (nomi simbolici) o numeri

Stringhe

È una sequenza di caratteri delimitata da doppi apici.

Variabili

Valori non specificati che possono essere determinati durante l'esecuzione delle query. Iniziano con una lettera maiuscola o con un «_»

Termino composto

È un atomo detto "funtore" con uno o più argomenti

Liste

Strutture dati fondamentali che rappresentano sequenze ordinate di elementi.

Esempio - Gestione Biblioteca

FUNTORE STRINGA ATOMO NUMERO ATOMO
~~~~~                    ~~~~~~                    ~~~~~~                    ~~~~~~                    ~~~~~~  
libro("Harry Potter", jk\_rowling, 1997, fantasy).

lista\_generi\_libri([fantasy, romantico, storico]).  
~~~~~  
FUNTORE LISTA

Componenti principali



Rappresentano esplicitamente
la conoscenza del dominio del
problema

Fatti

Proposizione che si presume vera

Un fatto è una dichiarazione che stabilisce una relazione tra gli elementi senza alcuna condizione o argomentazione.

Sono clausole di Horn non condizionali.



genitore (carlo, stefano)

Carlo è il genitore di Stefano

uomo (carlo)

Carlo è un uomo

Regole

Dichiarazione che specifica una relazione logica tra gli elementi basata su condizioni o inferenze.



conclusione

figlio (X,Y)

condizione

genitore (Y,X)

"Se Y è genitore di X, allora X è figlio di Y"

? **Query**

Consentono di esplorare le relazioni definite e di ottenere informazioni specifiche in modo interattivo

- Posta una domanda,
- Prolog risponde True o False per confermarne o meno la validità all'interno della base di conoscenza oppure restituisce il valore associato alla variabile.

? - genitore (carlo, stefano). **True.**

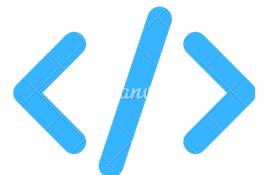
«È vero che Carlo è il genitore di Stefano?»

? - genitore (X, stefano). **X = carlo.**

«Chi è il genitore di Stefano?»

Variabili

Esprime i pronomi indefiniti "tutti" (nei fatti) e "qualcuno" (nelle regole)



```
vive (X, italia) :- abita (X, roma).
```

Possono essere usate nelle interrogazioni per farci dire per quali oggetti la domanda è soddisfatta.



```
? - genitore (X, francesco).
```

```
X = pietro
```

Operatori aritmetici

Essi sono utilizzati per eseguire calcoli matematici.

Le espressioni devono essere esplicitamente valutate utilizzando l'operatore **is**

Operatori di base :

+ - * / ^ mod

Operatori per confrontare
valori numerici:

=:= =\= > < =< >=

Esempio:

```
?- X is 3 + 2.  
% X = 5
```

Esempio:

```
?- 5 =:= 2 + 3.  
% true
```

Liste

Una lista è una sequenza ordinata di elementi

[Elemento1, Elemento2, ..., ElementoN]

Esempi:

[]

[a, b, c]

[1, 2, 3, 4]

[1, 'a', [2, 3]]

[[1, 2], [3, 4], [5, 6]]

Operatori principali



Cons (|)

È usato per dividere una lista in testa e coda.

Esempio:

[1, 2, 3]

Nella forma [H|T]

diventa:

[1 | [2, 3]]



Unificazione

È usato per decomporre le liste in testa e coda.

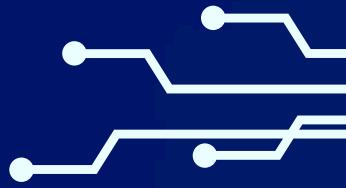
Esempio:

[First, Second | Tail] =

[a,b,c,d,e,f].

First = a, Second = b,

Tail = [c, d, e, f].



Predicati per operare sulle liste

member: Verifica se un elemento è presente in una lista.

```
member(X, [1, 2, 3]). % X può essere 1, 2 o 3.
```

append: (Lista1, Lista2, ListaConcatenata)

```
append([1, 2], [3, 4], L). % L sarà [1, 2, 3, 4]
```

```
append(X, Y, [1, 2, 3, 4]). % X = [] Y = [1, 2, 3, 4]
```

length: length (lista, Lunghezza)

```
length([1, 2, 3], L). % L sarà 3
```





reverse: inverte l'ordine degli elementi in una lista

```
reverse([1, 2, 3], L). % L sarà [3, 2, 1]
```

select: seleziona e rimuove un elemento da una lista

```
select(2, [1, 2, 3], L). % L sarà [1, 3]
```

findall: colleziona tutti i possibili valori che soddisfano
una certa condizione in una lista

```
findall(Template, Goal, List)
```

```
findall(Figlio, genitore(Genitore, Figlio), Figli).
```



If then Else

Prolog che permette di esprimere una condizione attraverso il costrutto condizionale ->;

```
(Condizione ->  
AzioneSeVera ;  
AzioneSeFalsa)
```

Esempio:

```
(Eta < 18 ->  
Categoria = giovane ;  
Categoria = adulto)
```



Cut (!)

Viene utilizzato per interrompere il backtracking una volta che una condizione è stata soddisfatta

Esempio:

```
categorizza(Eta, giovane) :-  
    Eta < 18, !.
```

```
categorizza(_, adulto).
```

-
-
-
-
-
-



Forza il fallimento di
una condizione
specificata permettendo
il passaggio alla regola
successiva.

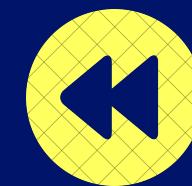
Paradigma

Prolog è un linguaggio di programmazione logica basato sul **paradigma dichiarativo**, che si concentra sulla dichiarazione di relazioni logiche invece di specificare istruzioni imperative, lasciando al sistema il compito di trovare la soluzione.

CONCETTI CHIAVE



Unificazione: Processo di corrispondenza tra termini e variabili.



Backtracking: Meccanismo di esplorazione delle alternative durante la ricerca delle soluzioni.



Ricorsione: permette di definire algoritmi iterativi e viene utilizzata per esplorare tutte le possibili soluzioni di un problema.



Inferenza: Il ragionamento avviene tramite l'applicazione di regole logiche e fatti per derivare nuove informazioni.

UNIFICAZIONE

L'unificazione è il processo mediante il quale Prolog tenta di rendere due termini identici, sia tramite confronto diretto sia sostituendo variabili con valori appropriati.

Regole

- S e T costanti: S e T sono unificabili se sono identiche.
- S è una variabile e T non è una variabile: sono unificabili con la sostituzione S=T
- S e T sono dei termini: sono unificabili solo se S e T hanno lo stesso funtore e tutti gli argomenti dei due termini sono a loro volta unificabili uno ad uno.



```
?- apple = apple.  
true.
```



```
?- X = apple.  
X = apple.
```

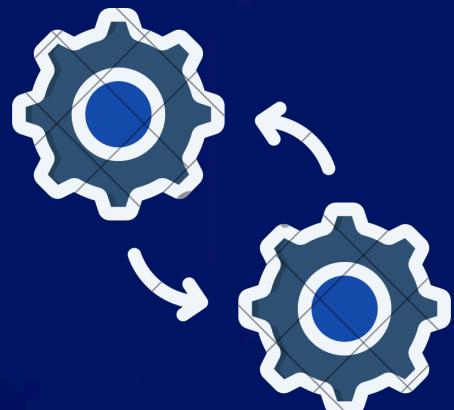


```
?- point(X, Y) = point(3, 4).  
X = 3, Y = 4.
```

UNIFICAZIONE

Concetto fondamentale: il sistema cerca di abbinare una query con fatti e regole della knowledge base, assegnando valori alle variabili per rendere la query vera.

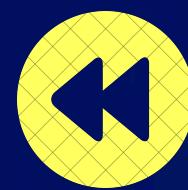
```
genitore(pietro, maria)  
? genitore(X, maria)
```



X = pietro

Prolog unificherà X con pietro per soddisfare la query.

BACKTRACKING E RICORSIONE



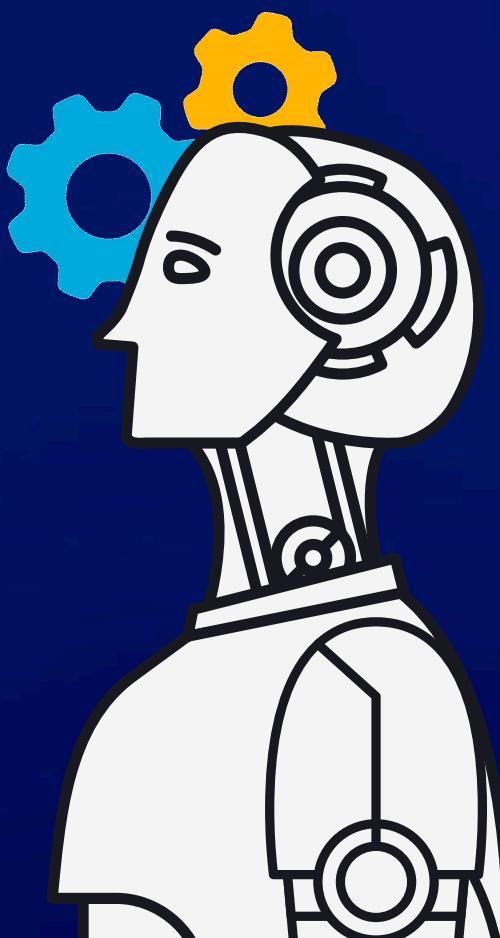
Il **backtracking** in Prolog permette al sistema di esplorare tutte le possibili soluzioni di una query tornando indietro e provando alternative quando un percorso fallisce.



La **ricorsione** è un concetto che consente a regole e predicati di richiamarsi direttamente o indirettamente, permettendo la definizione di algoritmi iterativi.

La ricorsione lavora insieme al backtracking in Prolog.

Le chiamate ricorsive possono essere utilizzate all'interno di un processo di backtracking per esplorare tutte le possibili soluzioni ad un problema.



INFERENZA

L'inferenza è il processo mediante il quale Prolog deriva nuove informazioni a partire da regole e fatti dichiarati nel programma.

Quando viene posta una domanda o una query a Prolog, il sistema utilizza l'inferenza per applicare le regole logiche definite nel programma e derivare le risposte corrispondenti.

Essa utilizza gli strumenti citati in precedenza come l'unificazione e il backtracking

Esempio:

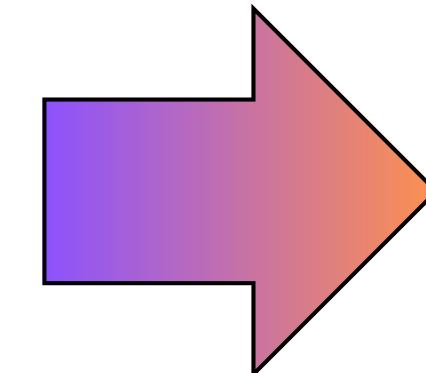
```
genitore(giovanni, maria).  
  
figlio(Y, X) :- genitore(X, Y).  
  
?- figlio(Figlio, giovanni).
```



Inteprete

L'interprete opera in due fasi:

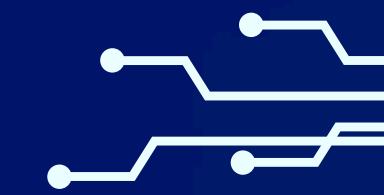
Memorizza le clausole che definiscono la conoscenza sul problema



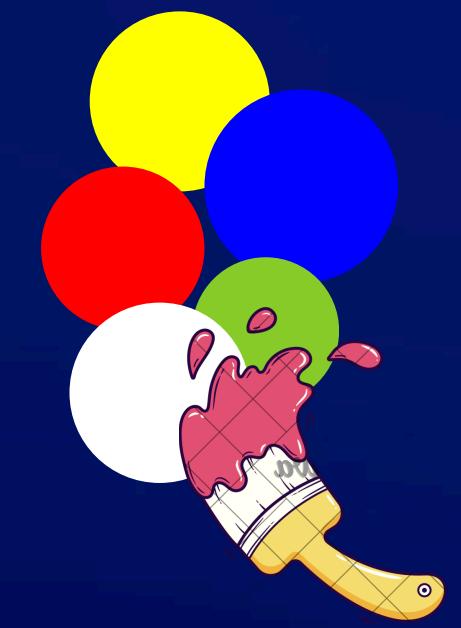
Risponde alle domande che permettono di risolvere il problema

Unificazione

Backtracking



Indovinello di Einstein



REGOLE INDOVINELLO

1. Si hanno cinque case colorate in fila, ciascuna con un proprietario, un animale domestico, delle sigarette e una bevanda.
 2. L'inglese vive nella casa rossa.
 3. Lo spagnolo ha un cane.
 4. Bevono caffè nella casa verde
 5. L'ucraino beve il tè.
 6. La casa verde è accanto alla casa bianca.
 7. Il fumatore Winston ha un serpente.
 8. Nella casa gialla fumano Kool.
 9. Nella casa di mezzo bevono il latte.
 10. Il norvegese abita nella prima casa da sinistra.
 11. Il fumatore di Chesterfield vive vicino all'uomo con la volpe.
 12. Nella casa vicino alla casa con il cavallo fumano Kool.
 13. Il fumatore Lucky Strike beve succo.
 14. Il giapponese fuma le Kent.
 15. Il norvegese vive vicino alla casa blu.
- Chi possiede la zebra?
- Chi beve l'acqua?
- 

Soluzione



Kool



Chesterfield



Winston



Kent



Lucky Strike

Implementazione codice

Struttura delle case:

house(Index, Nazionalità, Animale, Bevanda, Colore, Sigarette)

Definizione dei predicati che descrivono indizi:

```
● ● ●  
% 1) L'inglese vive in una casa rossa  
english_lives_in_red_house(Houses) :-  
    member(house(_, inglese, _, _, rosso, _), Houses).
```



Per essere vero il vincolo dovrà essere presente **house(_ , inglese, _ , _ , rosso , _)** nella lista Houses, dove i trattini indicano che i restanti attributi possono assumere qualsiasi altro valore.

Implementazione codice

```
% 6) la casa verde è vicino a quella bianca

green_next_to_white(Houses) :-  
    member(house(GreenIndex, _, _, _, verde, _), Houses),  
    LeftIndex is GreenIndex - 1,  
    RightIndex is GreenIndex + 1,  
    (member(house(LeftIndex, _, _, _, bianco, _), Houses);  
     member(house(RightIndex, _, _, _, bianco, _), Houses)).
```



Questo indizio esprime il **vincolo di vicinanza** determinando l'indice della casa verde e verificando che la casa bianca si trovi immediatamente a sinistra o a destra di essa.



Implementazione codice

Il predicato solve_houses definisce le case e applica tutti i vincoli.

```
% Risoluzione del puzzle  
solve_houses(Houses) :-  
  
% Inizializzazione delle Case:  
H1 = house(1, norvegese, _, _, _, _),  
H2 = house(2, _, _, _, _, _),  
H3 = house(3, _, _, latte, _, _),  
H4 = house(4, _, _, _, _, _),  
H5 = house(5, _, _, _, _, _),  
Houses = [H1, H2, H3, H4, H5],  
% Richiamo di tutti i vincoli
```

?- solve_houses(Houses).

Prolog tenterà di trovare una configurazione delle case (Houses) che soddisfi tutti i vincoli specificati.



Conclusioni

Vantaggi

- Logica dichiarativa
- Backtracking
- Facilità di modellazione delle conoscenze



Svantaggi

- Prestazioni
- Debugging complesso
- Limitazioni nella programmazione imperativa

