Prolog: materiale didattico

Qualunque testo di introduzione al Prolog, ad esempio:

- I. Bratko. Programmare in Prolog per l'Intelligenza Artificiale. Masson - Addison Wesley, 1988.
- W. F. Clocksin and C. S. Mellish. Programmare in Prolog. Franco Angeli, 1993.
- L. Console, E. Lamma, P. Mello, M. Milano. Programmazione Logica e Prolog. UTET Libreria, 1997.

Si può trovare materiale utile anche online.

Compilatore **SWI-Prolog**: `http://www.swi-prolog.org/`

Prolog: PROgramming in LOGic

Anni 70-80: ideato da Robert Kowalski; primo interprete Prolog (Alain Colmerauer, 1972); primo compilatore Prolog (Warren, 1983, tramite la Warren Abstract Machine).

È molto diverso dagli altri linguaggi di programmazione

- Dichiarativo (non procedurale)
- Ricorsione (niente cicli “for” o “while”)
- I costrutti di base sono relazioni (non comandi né funzioni)
- Concetti fondamentali: unificazione e backtracking

Un programma logico consiste di formule logiche e il calcolo è il processo di derivare conseguenze dal programma (costruzione di dimostrazioni)

Idea di base:

- Descrivere la situazione di interesse (programma)
- Fare una domanda (query)
- Il Prolog deduce nuovi fatti dal programma e risponde alla domanda

Programmare in Prolog

Un programma logico

- ha un significato **dichiarativo**: un programma consiste di fatti e regole che definiscono relazioni .
- ha un significato **operazionale**: meccanismo di esecuzione (motore di inferenza), basato sull'unificazione e la risoluzione, per estrarre conseguenze implicite nel programma

Per programmare in Prolog si devono avere presenti entrambi gli aspetti.

Conseguenze

- Cambiare modo di pensare: dichiarativamente, non proceduralmente.
- Linguaggio di alto livello
  - Non è efficiente quanto, ad esempio, il C
  - Ottimo per il "rapid prototyping"
  - Utile in molte applicazioni di Intelligenza Artificiale (rappresentazione della conoscenza, deduzione, analisi del linguaggio naturale...)

Programma Prolog: Fatti

```
/* Definizione del predicato amico
       (questo e' un commento) */
amico(antonio,bianca).
amico(antonio,enrico).
amico(bianca,dario).
amico(dario,carla).
amico(enrico,bianca).
```

`amico` è un **predicato**.
`antonio,bianca,carla,...` sono **costanti**.

Programma Prolog: Fatti

```
/* Definizione del predicato amico
       (questo e' un commento) */
amico(antonio,bianca).
amico(antonio,enrico).
amico(bianca,dario).
amico(dario,carla).
amico(enrico,bianca).
%%% antonio e` amico di tutti (questo e' un commento)
amico(antonio,Chiunque).
```

`amico` è un **predicato**.
`antonio,bianca,carla,...` sono **costanti**.
`Chiunque` è una **variabile**.

Programma Prolog: Fatti

```
/* Definizione del predicato amico
       (questo e' un commento) */
amico(antonio,bianca).
amico(antonio,enrico).
amico(bianca,dario).
amico(dario,carla).
amico(enrico,bianca).
%%% antonio e` amico di tutti (questo e' un commento)
amico(antonio,Chiunque).
```

`amico` è un **predicato**.
`antonio,bianca,carla,...` sono **costanti**.
`Chiunque` è una **variabile**.

Queries

```
?- amico(antonio,enrico).
true
```

Programma Prolog: Fatti

```
/* Definizione del predicato amico
       (questo e' un commento) */
amico(antonio,bianca).
amico(antonio,enrico).
amico(bianca,dario).
amico(dario,carla).
amico(enrico,bianca).
%%% antonio e` amico di tutti (questo e' un commento)
amico(antonio,Chiunque).
```

`amico` è un **predicato**.
`antonio,bianca,carla,...` sono **costanti**.
`Chiunque` è una **variabile**.

Queries

```
?- amico(dario,bianca).
false.
```

Programma Prolog: Fatti

```
/* Definizione del predicato amico
       (questo e' un commento) */
amico(antonio,bianca).
amico(antonio,enrico).
amico(bianca,dario).
amico(dario,carla).
amico(enrico,bianca).
%%% antonio e` amico di tutti (questo e' un commento)
amico(antonio,Chiunque).
```

`amico` è un **predicato**.
`antonio,bianca,carla,...` sono **costanti**.
`Chiunque` è una **variabile**.

Queries

```
?- amico(antonio,100).
true.
```

Programma Prolog: Regole

```
/* Definizione del predicato amico */
amico(antonio,bianca).
amico(antonio,enrico).
amico(bianca,dario).
amico(dario,carla).
amico(enrico,bianca).
amico(antonio,Chiunque).
/* Definizione del predicato conosce */
conosce(dario,enrico).
conosce(X,Y) :- amico(X,Y).
conosce(X,Y) :- amico(X,Qualcuno), amico(Qualcuno,Y).
```

`conosce(X,Y)` — **testa** della regola

`amico(X,Qualcuno), amico(Qualcuno,Y)` — **corpo** della regola

La virgola in Prolog rappresenta la congiunzione ($\wedge$)

Programma Prolog: Regole

```
/* Definizione del predicato amico */
amico(antonio,bianca).
amico(antonio,enrico).
amico(bianca,dario).
amico(dario,carla).
amico(enrico,bianca).
amico(antonio,Chiunque).
/* Definizione del predicato conosce */
conosce(dario,enrico).
conosce(X,Y) :- amico(X,Y).
conosce(X,Y) :- amico(X,Qualcuno), amico(Qualcuno,Y).
```

```
conosce(X,Y)
```
testa della regola

```
amico(X,Qualcuno), amico(Qualcuno,Y)
```
corpo della regola

La virgola in Prolog rappresenta la congiunzione ($\wedge$)

```
?- conosce(bianca,dario).
true
```

Programma Prolog: Regole

```
/* Definizione del predicato amico */
amico(antonio,bianca).
amico(antonio,enrico).
amico(bianca,dario).
amico(dario,carla).
amico(enrico,bianca).
amico(antonio,Chiunque).
/* Definizione del predicato conosce */
conosce(dario,enrico).
conosce(X,Y) :- amico(X,Y).
conosce(X,Y) :- amico(X,Qualcuno), amico(Qualcuno,Y).
```

```
conosce(X,Y)
amico(X,Qualcuno), amico(Qualcuno,Y)
```

testa della regola

corpo della regola

La virgola in Prolog rappresenta la congiunzione ($\wedge$)

```
?- conosce(bianca,carla).
true.
```

Programma Prolog: Regole

```
/* Definizione del predicato amico */
amico(antonio,bianca).
amico(antonio,enrico).
amico(bianca,dario).
amico(dario,carla).
amico(enrico,bianca).
amico(antonio,Chiunque).
/* Definizione del predicato conosce */
conosce(dario,enrico).
conosce(X,Y) :- amico(X,Y).
conosce(X,Y) :- amico(X,Qualcuno), amico(Qualcuno,Y).
```

```
conosce(X,Y)
```
testa della regola

```
amico(X,Qualcuno), amico(Qualcuno,Y)
```
corpo della regola

La virgola in Prolog rappresenta la congiunzione ($\wedge$)

```
?- conosce(bianca,enrico).
false.
```

La disgiunzione in Prolog

```
/* Definizione del predicato amico */
amico(antonio,bianca).
amico(antonio,enrico).
amico(bianca,dario).
amico(dario,carla).
amico(enrico,bianca).
amico(antonio,Chiunque).
/* Definizione del predicato conosce */
conosce(dario,enrico).
conosce(X,Y) :- amico(X,Y) ; amico(Y,X).
conosce(X,Y) :- amico(X,Qualcuno), amico(Qualcuno,Y).
```

Il punto e virgola in Prolog rappresenta la disgiunzione ($\vee$)

La disgiunzione in Prolog

```
/* Definizione del predicato amico */
amico(antonio,bianca).
amico(antonio,enrico).
amico(bianca,dario).
amico(dario,carla).
amico(enrico,bianca).
amico(antonio,Chiunque).
/* Definizione del predicato conosce */
conosce(dario,enrico).
conosce(X,Y) :- amico(X,Y) ; amico(Y,X).
conosce(X,Y) :- amico(X,Qualcuno), amico(Qualcuno,Y).
```

Il punto e virgola in Prolog rappresenta la disgiunzione ($\vee$)

```
?- conosce(bianca,enrico).
true.
```

Generazione di valori di risposta

Torniamo al programma senza disgiunzione

```
%% amico
amico(antonio,bianca).
amico(antonio,enrico).
amico(bianca,dario).
amico(dario,carla).
amico(enrico,bianca).
amico(antonio,Chiunque).
%% conosce
conosce(dario,enrico).
conosce(X,Y) :- amico(X,Y).
conosce(X,Y) :- amico(X,Qualcuno), amico(Qualcuno,Y).
```

Queries con variabili

```
?- conosce(bianca,X).
X = dario
```

Generazione di valori di risposta

Torniamo al programma senza disgiunzione

```
%% amico
amico(antonio,bianca).
amico(antonio,enrico).
amico(bianca,dario).
amico(dario,carla).
amico(enrico,bianca).
amico(antonio,Chiunque).
%% conosce
conosce(dario,enrico).
conosce(X,Y) :- amico(X,Y).
conosce(X,Y) :- amico(X,Qualcuno), amico(Qualcuno,Y).
```

Queries con variabili

```
?- conosce(bianca,X).
X = dario ;
```

Generazione di valori di risposta

Torniamo al programma senza disgiunzione

```
%% amico
amico(antonio,bianca).
amico(antonio,enrico).
amico(bianca,dario).
amico(dario,carla).
amico(enrico,bianca).
amico(antonio,Chiunque).
%% conosce
conosce(dario,enrico).
conosce(X,Y) :- amico(X,Y).
conosce(X,Y) :- amico(X,Qualcuno), amico(Qualcuno,Y).
```

Queries con variabili

```
?- conosce(bianca,X).
X = dario ;
X = carla.
```

Prolog e logica

Fatti e regole sono **clausole**.

	Prolog	Logica
Implicazione	`A :- B`	$B \rightarrow A$
Congiunzione	`A , B`	$A \wedge B$
Disgiunzione	`A ; B`	$A \vee B$

Prolog e logica: le variabili

- Variabili in un **fatto**:

  `amico(antonio,X)`: antonio e' amico di tutti (per ogni X, antonio e' amico di X).

  `pred(c1,...,cn,X1,...,Xn)`
  $\forall x_1 \ldots \forall x_n \, pred(c_1, \ldots, c_n, x_1, \ldots, x_n)$

Prolog e logica: le variabili

- Variabili in un **fatto**:

  `amico(antonio,X)`: antonio e' amico di tutti (per ogni X, antonio e' amico di X).

  `pred(c1,...,cn,X1,...,Xn)`
  $\forall x_1 \ldots \forall x_n \, pred(c_1, \ldots, c_n, x_1, \ldots, x_n)$

- Variabili in una **regola**

  `conosce(X,Y) :- amico(X,Qualcuno), amico(Qualcuno,Y).`
  Per ogni X e per ogni Y,
  X conosce Y se esiste Qualcuno conosciuto da X che conosce Y.

  `A(X1,...,Xn) :- B(X1,...,Xn,Y1,...,Yn).`
  $\forall x_1 \ldots \forall x_n \, (\exists y_1 \ldots \exists y_n B(x_1, \ldots, x_n, y_1, \ldots, y_n) \rightarrow A(x_1, \ldots, x_n))$

Prolog e logica: le variabili

- Variabili in un **fatto**:

  `amico(antonio,X)`: antonio e' amico di tutti (per ogni X, antonio e' amico di X).

  `pred(c1,...,cn,X1,...,Xn)`

  $\forall x_1 \ldots \forall x_n \, pred(c_1, \ldots, c_n, x_1, \ldots, x_n)$

- Variabili in una **regola**

  `conosce(X,Y) :- amico(X,Qualcuno), amico(Qualcuno,Y).`

  Per ogni X e per ogni Y,
  X conosce Y se esiste Qualcuno conosciuto da X che conosce Y.

  `A(X1,...,Xn) :- B(X1,...,Xn,Y1,...,Yn).`

  $\forall x_1 \ldots \forall x_n \, (\exists y_1 \ldots \exists y_n B(x_1, \ldots, x_n, y_1, \ldots, y_n) \rightarrow A(x_1, \ldots, x_n))$

  $\forall x_1 \ldots \forall x_n \forall y_1 \ldots \forall y_n \, (B(x_1, \ldots, x_n, y_1, \ldots, y_n) \rightarrow A(x_1, \ldots, x_n))$

Prolog e logica: le variabili

- Variabili in un **fatto**:

  `amico(antonio,X)`: antonio e' amico di tutti (per ogni X, antonio e' amico di X).

  `pred(c1,...,cn,X1,...,Xn)`
  $\forall x_1 \ldots \forall x_n \, pred(c_1, \ldots, c_n, x_1, \ldots, x_n)$

- Variabili in una **regola**

  `conosce(X,Y) :- amico(X,Qualcuno), amico(Qualcuno,Y).`

  Per ogni X e per ogni Y,
  X conosce Y se esiste Qualcuno conosciuto da X che conosce Y.

  `A(X1,...,Xn) :- B(X1,...,Xn,Y1,...,Yn).`
  $\forall x_1 \ldots \forall x_n \, (\exists y_1 \ldots \exists y_n B(x_1, \ldots, x_n, y_1, \ldots, y_n) \rightarrow A(x_1, \ldots, x_n))$
  $\forall x_1 \ldots \forall x_n \forall y_1 \ldots \forall y_n \, (B(x_1, \ldots, x_n, y_1, \ldots, y_n) \rightarrow A(x_1, \ldots, x_n))$

- Variabili in una **query**

  `?- conosce(bianca,X).` esiste X conosciuto da bianca?
  `?- goal(X1,...,Xn).` $\exists x_1 \ldots \exists x_n \, goal(x_1 \ldots x_n)$
  e' derivabile dal programma?
  Se lo e', per quali valori di $x_1 \ldots x_n$?

Procedure

Premessa: un fatto si può considerare come una regola il cui corpo è **true**, e la testa è il fatto stesso:

amico(antonio,bianca). amico(antonio,bianca) :- true.

Sia **pred** un predicato a *n* argomenti e **P** un programma Prolog.

L'insieme di fatti e regole in **P** la cui testa ha la forma

pred(termine$_1$,...,termine$_n$)

costituisce la **procedura** **pred**.

```
colore(tavolo,bianco).
colore(penna,rosso).
colore(X,Y) :- vicino(X,Z), colore(Z,Y).
```

Le clausole di una procedura rappresentano modi alternativi di risolvere problemi della stessa forma (colore(...,...)).

Il motore di inferenza del Prolog

```
genitore(tommaso,francesca).
genitore(tommaso,vittorio).
genitore(francesca,linda).
genitore(vittorio,bianca).
nonno(X,Y) :- genitore(X,Z), genitore(Z,Y).
```

```
?- nonno(tommaso,bianca).
```

Goal
nonno(tommaso,bianca)

Il motore di inferenza del Prolog

genitore(tommaso,francesca).
genitore(tommaso,vittorio).
genitore(francesca,linda).
genitore(vittorio,bianca).
$\Longrightarrow$ nonno(X,Y) :- genitore(X,Z), genitore(Z,Y).

?- nonno(tommaso,bianca).

Legami	Goal
	nonno(tommaso,bianca)
X=tommaso, Y=bianca	genitore(X,Z), genitore(Z,Y).

Il motore di inferenza del Prolog

$\Longrightarrow$ genitore(tommaso,francesca).
genitore(tommaso,vittorio).
genitore(francesca,linda).
genitore(vittorio,bianca).
nonno(X,Y) :- genitore(X,Z), genitore(Z,Y).

?- nonno(tommaso,bianca).

Legami	Goal
	nonno(tommaso,bianca)
X=tommaso, Y=bianca	genitore(X,Z), genitore(Z,Y).
X=tommaso, Y=bianca, Z=francesca	genitore(Z,Y): false

Il motore di inferenza del Prolog

genitore(tommaso,francesca).
$\Longrightarrow$ genitore(tommaso,vittorio).
genitore(francesca,linda).
genitore(vittorio,bianca).
nonno(X,Y) :- genitore(X,Z), genitore(Z,Y).

?- nonno(tommaso,bianca).

	Legami	Goal
		nonno(tommaso,bianca)
	X=tommaso, Y=bianca	genitore(X,Z), genitore(Z,Y).
	X=tommaso, Y=bianca, Z=francesca	genitore(Z,Y): false
Backtrack:	X=tommaso, Y=bianca, Z=vittorio	genitore(Z,Y)

Il motore di inferenza del Prolog

genitore(tommaso,francesca).
genitore(tommaso,vittorio).
genitore(francesca,linda).
$\Longrightarrow$ genitore(vittorio,bianca).
nonno(X,Y) :- genitore(X,Z), genitore(Z,Y).

?- nonno(tommaso,bianca).
true.

	Legami	Goal
		nonno(tommaso,bianca)
	X=tommaso, Y=bianca	genitore(X,Z), genitore(Z,Y).
	X=tommaso, Y=bianca, Z=francesca	genitore(Z,Y): false
Backtrack:	X=tommaso, Y=bianca, Z=vittorio	genitore(Z,Y)
	X=tommaso, Y=bianca, Z=vittorio	true

Il motore di inferenza del Prolog

```
genitore(tommaso,francesca).
genitore(tommaso,vittorio).
genitore(francesca,linda).
genitore(vittorio,bianca).
nonno(X,Y) :- genitore(X,Z), genitore(Z,Y).
```

```
?- nonno(tommaso,Chi).
```

Goal
nonno(tommaso,Chi)

Il motore di inferenza del Prolog

genitore(tommaso,francesca).
genitore(tommaso,vittorio).
genitore(francesca,linda).
genitore(vittorio,bianca).
$\Longrightarrow$ nonno(X,Y) :- genitore(X,Z), genitore(Z,Y).

?- nonno(tommaso,Chi).

Legami	Goal
	nonno(tommaso,Chi)
X=tommaso,Chi=Y	genitore(X,Z), genitore(Z,Y).

Il motore di inferenza del Prolog

$\Longrightarrow$ genitore(tommaso,francesca).
genitore(tommaso,vittorio).
genitore(francesca,linda).
genitore(vittorio,bianca).
nonno(X,Y) :- genitore(X,Z), genitore(Z,Y).

?- nonno(tommaso,Chi).

Legami	Goal
	nonno(tommaso,Chi)
X=tommaso,Chi=Y	genitore(X,Z), genitore(Z,Y).
X=tommaso, Chi=Y, Z=francesca	genitore(Z,Y)

Il motore di inferenza del Prolog

genitore(tommaso,francesca).
genitore(tommaso,vittorio).
$\Longrightarrow$ genitore(francesca,linda).
genitore(vittorio,bianca).
nonno(X,Y) :- genitore(X,Z), genitore(Z,Y).

?- nonno(tommaso,Chi).
Chi=linda

Legami	Goal	Risposta
	nonno(tommaso,Chi)	
X=tommaso,Chi=Y	genitore(X,Z), genitore(Z,Y).	
X=tommaso, Chi=Y, Z=francesca	genitore(Z,Y)	
X=tommaso, Chi=Y=linda, Z=francesca	true	Chi=linda

Il motore di inferenza del Prolog

```
genitore(tommaso,francesca).
genitore(tommaso,vittorio).
genitore(francesca,linda).
genitore(vittorio,bianca).
nonno(X,Y) :- genitore(X,Z), genitore(Z,Y).
```

```
?- nonno(tommaso,Chi).
Chi=linda ; ⟸
```

Legami	Goal	Risposta
	nonno(tommaso,Chi)	
X=tommaso,Chi=Y	genitore(X,Z), genitore(Z,Y).	
X=tommaso, Chi=Y, Z=francesca	genitore(Z,Y)	
X=tommaso, Chi=Y=linda, Z=francesca	true	Chi=linda
Backtrack: X=tommaso,Chi=Y	genitore(X,Z), genitore(Z,Y).	

Il motore di inferenza del Prolog

genitore(tommaso,francesca).
$\Longrightarrow$ genitore(tommaso,vittorio).
genitore(francesca,linda).
genitore(vittorio,bianca).
nonno(X,Y) :- genitore(X,Z), genitore(Z,Y).

?- nonno(tommaso,Chi).
Chi=linda ;

Legami	Goal	Risposta
	nonno(tommaso,Chi)	
X=tommaso,Chi=Y	genitore(X,Z), genitore(Z,Y).	
X=tommaso, Chi=Y, Z=francesca	genitore(Z,Y)	
X=tommaso, Chi=Y=linda, Z=francesca	true	Chi=linda
Backtrack: X=tommaso,Chi=Y	genitore(X,Z), genitore(Z,Y).	
X=tommaso, Chi=Y, Z=vittorio	genitore(Z,Y)	

Il motore di inferenza del Prolog

genitore(tommaso,francesca).
genitore(tommaso,vittorio).
genitore(francesca,linda).
$\Longrightarrow$ genitore(vittorio,bianca).
nonno(X,Y) :- genitore(X,Z), genitore(Z,Y).

?- nonno(tommaso,Chi).
Chi=linda ;
Chi=bianca

Legami	Goal	Risposta
	nonno(tommaso,Chi)	
X=tommaso,Chi=Y	genitore(X,Z), genitore(Z,Y).	
X=tommaso, Chi=Y, Z=francesca	genitore(Z,Y)	
X=tommaso, Chi=Y=linda, Z=francesca	true	Chi=linda
Backtrack: X=tommaso,Chi=Y	genitore(X,Z), genitore(Z,Y).	
X=tommaso, Chi=Y, Z=vittorio	genitore(Z,Y)	
X=tommaso, Chi=Y=bianca, Z=vittorio	true	Chi=bianca

Il backtracking

```
genitore(tommaso,francesca).
genitore(tommaso,vittorio).
genitore(francesca,linda).
genitore(francesca,leonardo).
genitore(vittorio,bianca).
genitore(vittorio,andrea).
nonno(X,Y) :- genitore(X,Z), genitore(Z,Y).

?- nonno(tommaso,Chi).
```

Il backtracking

```
genitore(tommaso,francesca).
genitore(tommaso,vittorio).
genitore(francesca,linda).
genitore(francesca,leonardo).
genitore(vittorio,bianca).
genitore(vittorio,andrea).
nonno(X,Y) :- genitore(X,Z), genitore(Z,Y).

?- nonno(tommaso,Chi).
Chi = linda
```

Il backtracking

```
genitore(tommaso,francesca).
genitore(tommaso,vittorio).
genitore(francesca,linda).
genitore(francesca,leonardo).
genitore(vittorio,bianca).
genitore(vittorio,andrea).
nonno(X,Y) :- genitore(X,Z), genitore(Z,Y).

?- nonno(tommaso,Chi).
Chi = linda ;
Chi = leonardo
```

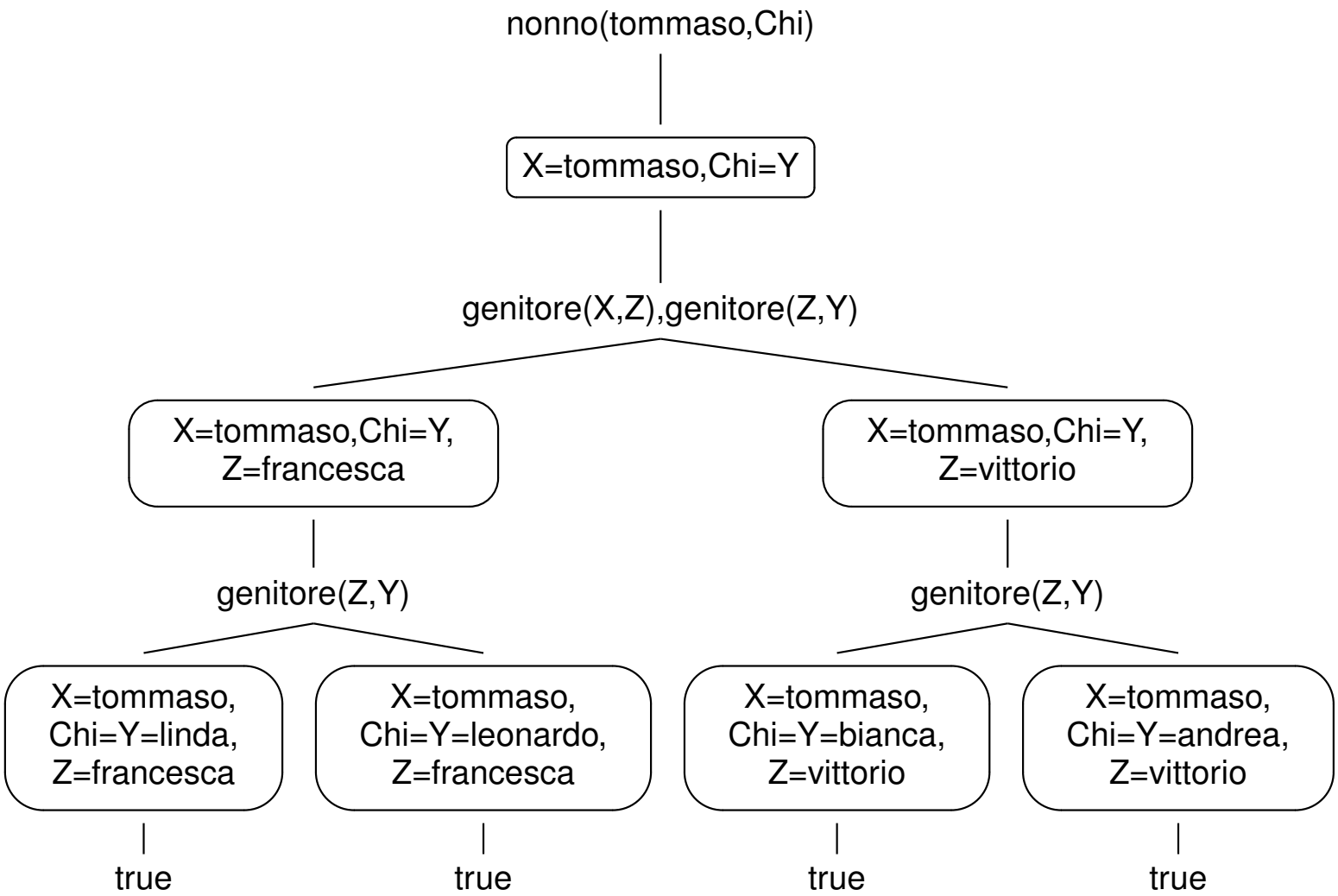
Il backtracking

```
genitore(tommaso,francesca).
genitore(tommaso,vittorio).
genitore(francesca,linda).
genitore(francesca,leonardo).
genitore(vittorio,bianca).
genitore(vittorio,andrea).
nonno(X,Y) :- genitore(X,Z), genitore(Z,Y).

?- nonno(tommaso,Chi).
Chi = linda ;
Chi = leonardo ;
Chi = bianca
```

Il backtracking

```
genitore(tommaso,francesca).
genitore(tommaso,vittorio).
genitore(francesca,linda).
genitore(francesca,leonardo).
genitore(vittorio,bianca).
genitore(vittorio,andrea).
nonno(X,Y) :- genitore(X,Z), genitore(Z,Y).

?- nonno(tommaso,Chi).
Chi = linda ;
Chi = leonardo ;
Chi = bianca ;
Chi = andrea.
```

L'albero di ricerca



Il backtracking

```
likes(mary,food).    % clausola 1
likes(mary,wine).    % clausola 2
likes(john,beer).    % clausola 3
likes(john,wine).    % clausola 4
likes(john,mary).    % clausola 5

?- likes(mary,X), likes(john,X).
```

Goal: likes(mary,X), likes(john,X).

Il backtracking

```
likes(mary,food).    % clausola 1
likes(mary,wine).    % clausola 2
likes(john,beer).    % clausola 3
likes(john,wine).    % clausola 4
likes(john,mary).    % clausola 5

?- likes(mary,X), likes(john,X).
```

Goal: likes(mary,X), likes(john,X).

- Sottogoal 1: likes(mary,X).
  Clausola usata: 1, legami: X=food
  Puntatore alla prossima clausola: 2.

Il backtracking

```
likes(mary,food).    % clausola 1
likes(mary,wine).    % clausola 2
likes(john,beer).    % clausola 3
likes(john,wine).    % clausola 4
likes(john,mary).    % clausola 5

?- likes(mary,X), likes(john,X).
```

Goal: likes(mary,X), likes(john,X).

- Sottogoal 1: likes(mary,X).
  Clausola usata: 1, legami: X=food
  Puntatore alla prossima clausola: 2.

- Sottogoal 2: likes(john,food).

Il backtracking

```
likes(mary,food).    % clausola 1
likes(mary,wine).    % clausola 2
likes(john,beer).    % clausola 3
likes(john,wine).    % clausola 4
likes(john,mary).    % clausola 5

?- likes(mary,X), likes(john,X).
```

Goal: likes(mary,X), likes(john,X).

- Sottogoal 1: likes(mary,X).
  Clausola usata: 1, legami: X=food
  Puntatore alla prossima clausola: 2.

- Sottogoal 2: likes(john,food).

Il backtracking

```
likes(mary,food).    % clausola 1
likes(mary,wine).    % clausola 2
likes(john,beer).    % clausola 3
likes(john,wine).    % clausola 4
likes(john,mary).    % clausola 5

?- likes(mary,X), likes(john,X).
```

Goal: likes(mary,X), likes(john,X).

- Sottogoal 1: likes(mary,X).
  Clausola usata: 1, legami: X=food
  Puntatore alla prossima clausola: 2.

- Sottogoal 2: likes(john,food).
  false.
  Backtrack.

Il backtracking

```
likes(mary,food).    % clausola 1
likes(mary,wine).    % clausola 2
likes(john,beer).    % clausola 3
likes(john,wine).    % clausola 4
likes(john,mary).    % clausola 5

?- likes(mary,X), likes(john,X).
```

Goal: likes(mary,X), likes(john,X).

- Sottogoal 1: likes(mary,X).
  Clausola usata: 1, legami: X=food
  Puntatore alla prossima clausola: 2.
  Backtracking: Clausola usata: 2, legami: X=wine
- Sottogoal 2: likes(john,food).
  false.
  Backtrack.

Il backtracking

```
likes(mary,food).    % clausola 1
likes(mary,wine).    % clausola 2
likes(john,beer).    % clausola 3
likes(john,wine).    % clausola 4
likes(john,mary).    % clausola 5

?- likes(mary,X), likes(john,X).
```

Goal: likes(mary,X), likes(john,X).

- Sottogoal 1: likes(mary,X).
  Clausola usata: 1, legami: X=food
  Puntatore alla prossima clausola: 2.
  Backtracking: Clausola usata: 2, legami: X=wine
- Sottogoal 2: likes(john,food).
  false.
  Backtrack.
  Sottogoal 2: likes(john,wine).

Il backtracking

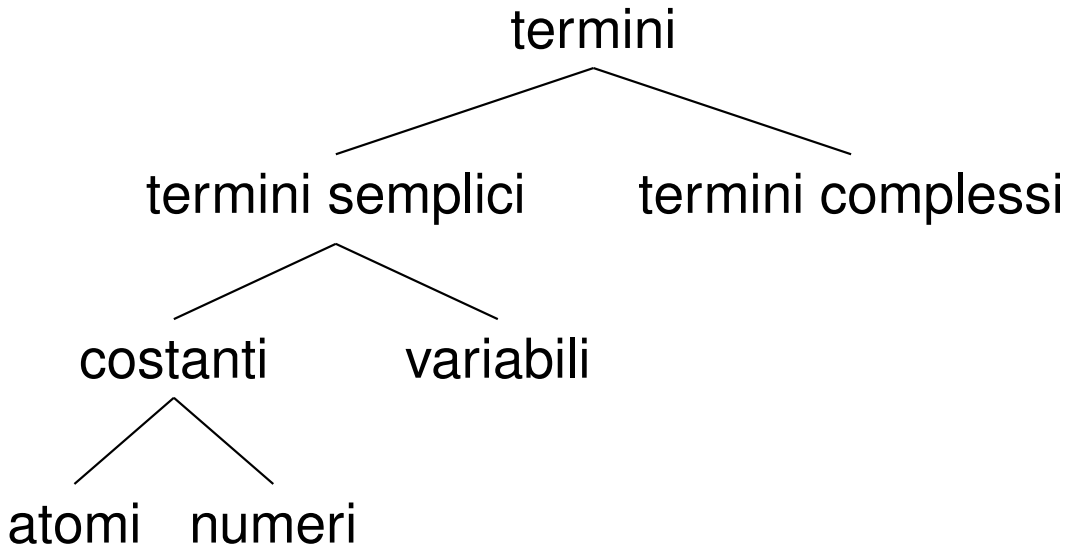
```
likes(mary,food).    % clausola 1
likes(mary,wine).    % clausola 2
likes(john,beer).    % clausola 3
likes(john,wine).    % clausola 4
likes(john,mary).    % clausola 5

?- likes(mary,X), likes(john,X).
```

Goal: likes(mary,X), likes(john,X).

- Sottogoal 1: likes(mary,X).
  Clausola usata: 1, legami: X=food
  Puntatore alla prossima clausola: 2.
  Backtracking: Clausola usata: 2, legami: X=wine
- Sottogoal 2: likes(john,food).
  false.
  Backtrack.
  Sottogoal 2: likes(john,wine).
  true: X=wine

I termini Prolog

Fatti, regole e queries sono costruiti sulla base di termini



- Atomi: tommaso, toMMaso_27, 'Tommaso d\'Aquino'.
  Sequenze di caratteri speciali: `: , ; . :-`
- Numeri: 12, -34, 345.01, 0.328
- Variabili: iniziano con lettera maiuscola o *underscore*: X, Y, Chi, _variabile

Classificazione dei termini

Il Prolog è un linguaggio a **tipizzazione dinamica**

Predicati predefiniti per riconoscere il tipo di un termine

var, nonvar

```
?- var(X).
true.

?- var(padre(X)).
false.

?- X=Y, Y=23, var(X).
false.
```

integer, float, number
atom, atomic

```
?- atom(pippo).
true.

?- atom(3).
false.

?- atomic(3).
true.
```

Termini complessi (o strutture)

Sono costruiti da un funtore (atomo) applicato a una sequenza di argomenti (altri termini, semplici o complessi):

$$\textbf{funtore(termine}_1\textbf{,...,termine}_n\textbf{)}$$

Esempi:

- libro('Le tigri di Mompracem',autore(emilio,salgari))
- padre(padre(antonio))
- 3+2

Termini complessi (o strutture)

Sono costruiti da un funtore (atomo) applicato a una sequenza di argomenti (altri termini, semplici o complessi):

$$\textbf{funtore(termine}_1\textbf{,...,termine}_n\textbf{)}$$

Esempi:

- libro('Le tigri di Mompracem',autore(emilio,salgari))
- padre(padre(antonio))
- 3+2
- nonno(X,maria)
- antenato(padre(padre(antonio)),madre(clara))

anche i fatti sono strutture

Termini complessi (o strutture)

Sono costruiti da un funtore (atomo) applicato a una sequenza di argomenti (altri termini, semplici o complessi):

$$\textbf{funtore(termine}_1\textbf{,...,termine}_n\textbf{)}$$

Esempi:

- libro('Le tigri di Mompracem',autore(emilio,salgari))
- padre(padre(antonio))
- 3+2
- nonno(X,maria)
- antenato(padre(padre(antonio)),madre(clara))

  anche i fatti sono strutture
- genitore(X,Y), genitore(Y,Z)

  anche una sequenza di goal è una struttura

  la virgola è l'operatore funtore principale

Termini complessi (o strutture)

Sono costruiti da un funtore (atomo) applicato a una sequenza di argomenti (altri termini, semplici o complessi):

$$\textbf{funtore(termine}_1\textbf{,...,termine}_n\textbf{)}$$

Esempi:

- libro('Le tigri di Mompracem',autore(emilio,salgari))
- padre(padre(antonio))
- 3+2
- nonno(X,maria)
- antenato(padre(padre(antonio)),madre(clara))
  anche i fatti sono strutture
- genitore(X,Y), genitore(Y,Z)
  anche una sequenza di goal è una struttura
  la virgola è l'operatore funtore principale
- figlio(X,Y) :- genitore(Y,X)
  anche una clausola è una struttura
  l'operatore principale è :-

Non c'è differenza sintattica tra fatti, regole, goal e strutture che possono essere argomenti di un funtore.

Unificazione

Due termini sono unificabili se:

- sono lo stesso termine, oppure
- contengono variabili che possono essere istanziate (sostituite) con termini in modo tale da rendere uguali i due termini

Unificazione

Due termini sono unificabili se:

- sono lo stesso termine, oppure
- contengono variabili che possono essere istanziate (sostituite) con termini in modo tale da rendere uguali i due termini

Unificabili		
termini		sostituzione
tommaso	tommaso	
tommaso	X	X=tommaso
nonno(X,Y)	nonno(tommaso,Z)	X=tommaso, Y=Z
nonno(tommaso,X)	nonno(Y,linda)	Y=tommaso, X=linda

Unificazione

Due termini sono unificabili se:

- sono lo stesso termine, oppure
- contengono variabili che possono essere istanziate (sostituite) con termini in modo tale da rendere uguali i due termini

Unificabili		
termini		sostituzione
tommaso	tommaso	
tommaso	X	X=tommaso
nonno(X,Y)	nonno(tommaso,Z)	X=tommaso, Y=Z
nonno(tommaso,X)	nonno(Y,linda)	Y=tommaso, X=linda

Non unificabili	
tommaso	vittorio
nonno(tommaso,X)	nonno(vittorio,linda)
nonno(tommaso,X)	nonno(X,linda)

Unificazione e uguaglianza

```
?- tommaso=tommaso.
true.
```

Unificazione e uguaglianza

```
?- tommaso=tommaso.
true.

?- tommaso=X.
X = tommaso.
```

Unificazione e uguaglianza

```
?- tommaso=tommaso.
true.

?- tommaso=X.
X = tommaso.

?- nonno(X,Y), nonno(tommaso,Z).
X = tommaso,
Y = linda,
Z = linda.
```

Unificazione e uguaglianza

```
?- tommaso=tommaso.
true.

?- tommaso=X.
X = tommaso.

?- nonno(X,Y), nonno(tommaso,Z).
X = tommaso,
Y = linda,
Z = linda.

?- nonno(X,Y) = nonno(tommaso,Z).
X = tommaso,
Y = Z.
```

Unificazione e uguaglianza

```
?- tommaso=tommaso.
true.

?- tommaso=X.
X = tommaso.

?- nonno(X,Y), nonno(tommaso,Z).
X = tommaso,
Y = linda,
Z = linda.

?- nonno(X,Y) = nonno(tommaso,Z).
X = tommaso,
Y = Z.

?- nonno(tommaso,X) = nonno(Y,linda).
X = linda,
Y = tommaso.
```

Unificazione e uguaglianza

```
?- nonno(tommaso,X) = nonno(X,linda).
false.
```

L'algortimo di unificazione sostituisce le variabili via via che trova come rendere uguali dei sottotermini

Unificazione e uguaglianza

```
?- nonno(tommaso,X) = nonno(X,linda).
false.
```

L'algortimo di unificazione sostituisce le variabili via via che trova come rendere uguali dei sottotermini

$$\begin{array}{l}\text{nonno(tommaso,X)}\\ \text{nonno(X,linda)}\end{array} \Longrightarrow \text{X=tommaso}$$

Unificazione e uguaglianza

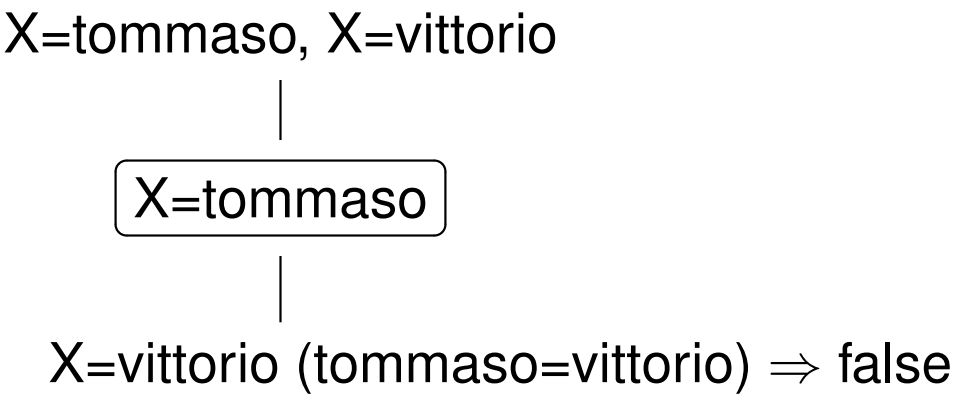
```
?- nonno(tommaso,X) = nonno(X,linda).
false.
```

L'algortimo di unificazione sostituisce le variabili via via che trova come rendere uguali dei sottotermini

nonno(tommaso,tommaso)
nonno(tommaso,linda)
$\Longrightarrow$ X=tommaso, NO

```
?- X=tommaso, X=vittorio.
false.
```

Quando due termini vengono unificati, il Prolog applica le sostituzioni dappertutto (fino alla fine del goal).



Algoritmo di unificazione (intuizione)

p(**X**,f(c),X)
p(**f(Y)**,Y,Z) $\Longrightarrow$ X=f(Y)

nuovo problema:

p(**f(Y)**,f(c),**f(Y)**)
p(f(Y),Y,Z)

Algoritmo di unificazione (intuizione)

p(**X**,f(c),X)
p(**f(Y)**,Y,Z) $\Longrightarrow$ X=f(Y)

nuovo problema:

p(**f(Y)**,f(c),**f(Y)**)
p(f(Y),Y,Z)

p(f(Y),**f(c)**,f(Y))
p(f(Y),**Y**,Z) $\Longrightarrow$ Y=f(c), X=f(**f(c)**)

nuovo problema:

p(f(**f(c)**),f(c),f(**f(c)**))
p(f(**f(c)**),**f(c)**,Z)

Algoritmo di unificazione (intuizione)

p(**X**,f(c),X)
p(**f(Y)**,Y,Z) $\Longrightarrow$ X=f(Y)

nuovo problema:

p(**f(Y)**,f(c),**f(Y)**)
p(f(Y),Y,Z)

p(f(Y),**f(c)**,f(Y))
p(f(Y),**Y**,Z) $\Longrightarrow$ Y=f(c), X=f(**f(c)**)

nuovo problema:

p(f(**f(c)**),f(c),f(**f(c)**))
p(f(**f(c)**),**f(c)**,Z)

p(f(f(c)),f(c),**f(f(c))**)
p(f(f(c)),f(c),**Z**) $\Longrightarrow$ Z=f(f(c)), Y=f(c), X=f(f(c))

Algoritmo di unificazione (intuizione)

p(**X**,f(c),X)
p(**f(Y)**,Y,Z) $\Longrightarrow$ X=f(Y)

nuovo problema:

p(**f(Y)**,f(c),**f(Y)**)
p(f(Y),Y,Z)

p(f(Y),**f(c)**,f(Y))
p(f(Y),**Y**,Z) $\Longrightarrow$ Y=f(c), X=f(**f(c)**)

nuovo problema:

p(f(**f(c)**),f(c),f(**f(c)**))
p(f(**f(c)**),**f(c)**,Z)

p(f(f(c)),f(c),**f(f(c))**)
p(f(f(c)),f(c),**Z**) $\Longrightarrow$ Z=f(f(c)), Y=f(c), X=f(f(c))

nuovo problema:

p(f(f(c)),f(c),f(f(c)))
p(f(f(c)),f(c),**f(f(c))**)

Occur check

X e **padre(X)** sono unificabili?

Occur check

X e **padre(X)** sono unificabili?

X
padre(X) $\Longrightarrow$ X=padre(X) ?

nuovo problema:

padre(X)
padre(**padre(X)**)

padre(**X**)
padre(**padre(X)**) $\Longrightarrow$ X=padre(padre(X)) ?

Occur check: è possibile unificare una variabile X con un termine solo se X non occorre nel termine.

Occur check

X e **padre(X)** sono unificabili?

X
padre(X) $\Longrightarrow$ X=padre(X) ?

nuovo problema:

padre(X)
padre(**padre(X)**)

padre(**X**)
padre(**padre(X)**) $\Longrightarrow$ X=padre(padre(X)) ?

Occur check: è possibile unificare una variabile X con un termine solo se X non occorre nel termine.

Ma l'unificazione del Prolog non esegue l'occur check:

```
?- X = padre(X).
X=padre(padre(padre(...))).
```

Occur check

X e **padre(X)** sono unificabili?

X
padre(X) $\Longrightarrow$ X=padre(X) ?

nuovo problema:

padre(X)
padre(**padre(X)**)

padre(**X**)
padre(**padre(X)**) $\Longrightarrow$ X=padre(padre(X)) ?

Occur check: è possibile unificare una variabile X con un termine solo se X non occorre nel termine.

Ma l'unificazione del Prolog non esegue l'occur check:

```
?- X = padre(X).
X = padre(**).
```

Occur check

X e **padre(X)** sono unificabili?

X
padre(X) $\Longrightarrow$ X=padre(X) ?

nuovo problema:

padre(X)
padre(**padre(X)**)

padre(**X**)
padre(**padre(X)**) $\Longrightarrow$ X=padre(padre(X)) ?

Occur check: è possibile unificare una variabile X con un termine solo se X non occorre nel termine.

Ma l'unificazione del Prolog non esegue l'occur check:

```
?- X = padre(X).
X = padre(**).

?- unify_with_occurs_check(X,padre(X)).
false.
```

Programmare con l'unificazione

```
vertical(line(point(X,Y),point(X,Z))).
horizontal(line(point(X,Y),point(Z,Y))).
```

Programmare con l'unificazione

```
vertical(line(point(X,Y),point(X,Z))).
horizontal(line(point(X,Y),point(Z,Y))).

?- vertical(line(point(1,1),point(1,3))).
true.
```

Programmare con l'unificazione

```
vertical(line(point(X,Y),point(X,Z))).
horizontal(line(point(X,Y),point(Z,Y))).

?- vertical(line(point(1,1),point(1,3))).
true.

?- vertical(line(point(1,1),point(3,2))).
false.
```

Programmare con l'unificazione

```
vertical(line(point(X,Y),point(X,Z))).
horizontal(line(point(X,Y),point(Z,Y))).

?- vertical(line(point(1,1),point(1,3))).
true.

?- vertical(line(point(1,1),point(3,2))).
false.

?- horizontal(line(point(1,1),point(1,Y))).
Y = 1.
```

Programmare con l'unificazione

```
vertical(line(point(X,Y),point(X,Z))).
horizontal(line(point(X,Y),point(Z,Y))).

?- vertical(line(point(1,1),point(1,3))).
true.

?- vertical(line(point(1,1),point(3,2))).
false.

?- horizontal(line(point(1,1),point(1,Y))).
Y = 1.

?- horizontal(line(point(2,3),Point)).
Point = point(_G650, 3).
```

L'aritmetica in Prolog

```
?- 3+2 = X.
X = 3+2.
```

```
?- 3+2 = 5.
false.
```

L'aritmetica in Prolog

```
?- 3+2 = X.
X = 3+2.

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

?- 5 is 3+2.
true.

?- X is 3.0 + 2.0.
X = 5.0.

?- X is 3.0 + 2.
X = 5.0.
```

```
?- 3+2 = 5.
false.

/* is e non = */
```

L'aritmetica in Prolog

```
?- 3+2 = X.
X = 3+2.

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

?- 5 is 3+2.
true.

?- X is 3.0 + 2.0.
X = 5.0.

?- X is 3.0 + 2.
X = 5.0.

?- 5 is 3+X.
ERROR: is/2: Arguments are not sufficiently instantiated
```

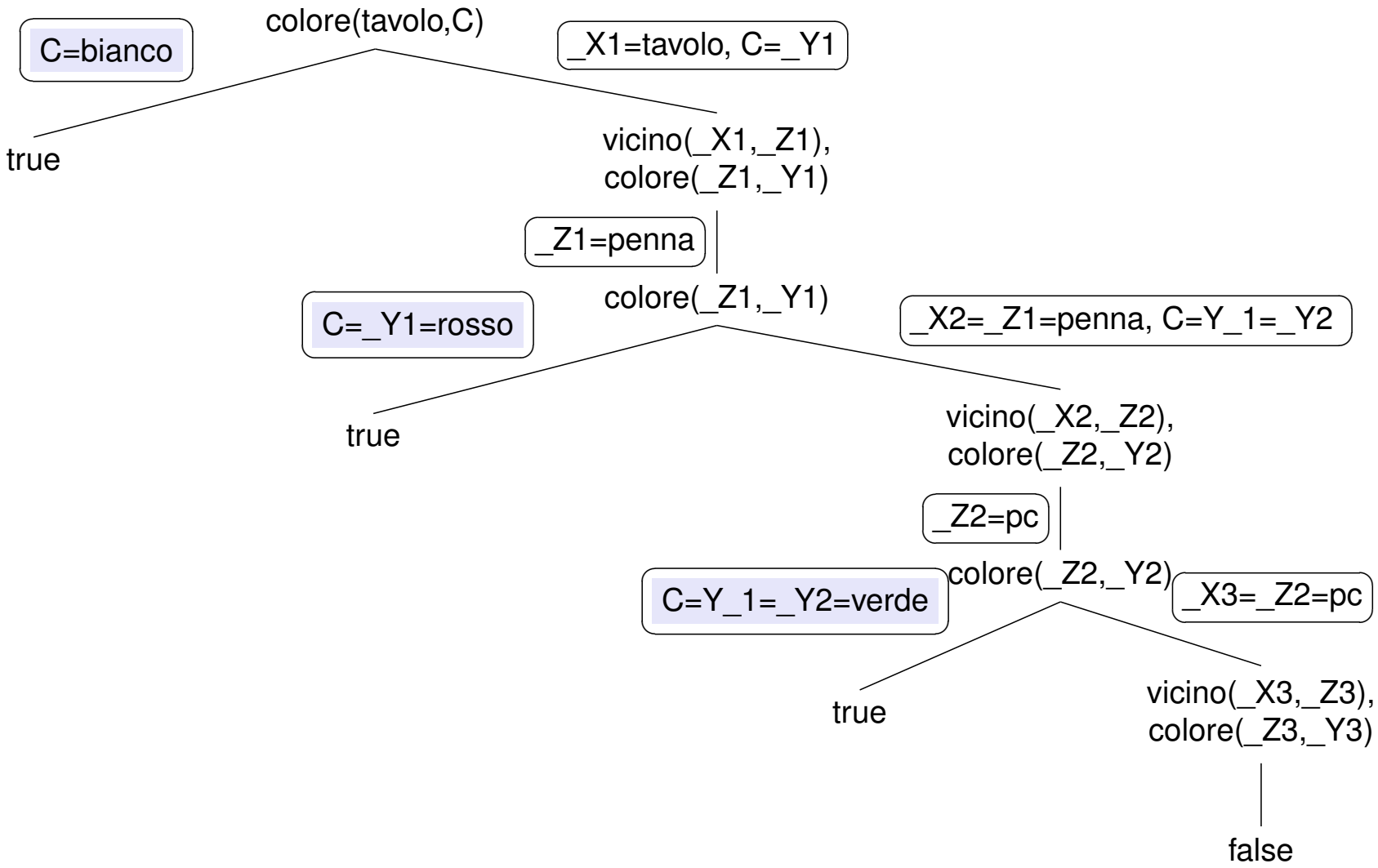
```
?- 3+2 = 5.
false.

/* is e non = */
```

Procedure ricorsive

```
vicino(tavolo,penna).
vicino(penna,pc).

colore(tavolo,bianco).
colore(penna,rosso).
colore(pc,verde).
colore(X,Y) :- vicino(X,Z), colore(Z,Y).

?- colore(tavolo,C).
C = bianco ;
C = rosso ;
C = verde ;
false.
```

L'abero di ricerca

```
vicino(tavolo,penna).   vicino(penna,pc).
colore(tavolo,bianco).  colore(penna,rosso).  colore(pc,verde).
colore(X,Y) :- vicino(X,Z), colore(Z,Y).
```



Reversibilità dei programmi

```
vicino(tavolo,penna).   vicino(penna,pc).
colore(tavolo,bianco).  colore(penna,rosso).  colore(pc,verde).
colore(X,Y) :- vicino(X,Z), colore(Z,Y).
```

```
?- colore(X,rosso).
X = penna ;
X = tavolo ;
false.
```

```
?- colore(X,Y).
X = tavolo,
Y = bianco ;
X = penna,
Y = rosso ;
X = pc,
Y = verde ;
X = tavolo,
Y = rosso ;
..........
```

La strategia di ricerca del Prolog

La strategia di ricerca del Prolog visita l'albero di ricerca in profondità:

- considera le clausole del programma dall'alto in basso
- considera il corpo delle clausole da sinistra a destra
- torna indietro (backtrack) sulle scelte sbagliate

Il significato dichiarativo e quello procedurale di un programma possono non coincidere: il Prolog può espandere un cammino infinito e non trovare mai una soluzione, anche se una soluzione esiste.

È importante:

- l'ordine delle clausole nel programma
- l'ordine dei sottogoal nel corpo delle regole

Regole di programmazione:

- In una procedura: prima i fatti, poi le regole
- Nel corpo di una regola: prima i goal più facili da dimostrare o refutare, poi quelli più difficili