
Linguaggi di Programmazione I – Lezione 18

Proff. Piero Bonatti e Marco Faella

<mailto://pab@unina.it>

<mailto://marfaella@gmail.com>

27 maggio 2024



Panoramica dell'argomento

Paradigma logico

Prolog

Costrutti Prolog

Unificazione

Conjunctive queries

Le Regole

Ragionamento

Liste

Applicazioni

Programmazione nondeterministica

Unicità di Prolog



Paradigma logico

Prolog

Costrutti Prolog

Unificazione

Conjunctive queries

Le Regole

Ragionamento

Liste

Applicazioni

Programmazione
nondeterministica

Unicità di Prolog

Paradigma logico



L'essenza del paradigma logico

programmi = insiemi di assiomi

[Paradigma logico](#)

[Prolog](#)

[Costrutti Prolog](#)

[Unificazione](#)

[Conjunctive queries](#)

[Le Regole](#)

[Ragionamento](#)

[Liste](#)

[Applicazioni](#)

[Programmazione
nondeterministica](#)

[Unicità di Prolog](#)



L'essenza del paradigma logico

[Paradigma logico](#)

[Prolog](#)

[Costrutti Prolog](#)

[Unificazione](#)

[Conjunctive queries](#)

[Le Regole](#)

[Ragionamento](#)

[Liste](#)

[Applicazioni](#)

[Programmazione
nondeterministica](#)

[Unicità di Prolog](#)

programmi = insiemi di assiomi

computazioni = dimostrazioni costruttive di una formula logica
data – detta *query* – mediante gli assiomi del programma



L'essenza del paradigma logico

[Paradigma logico](#)

[Prolog](#)

[Costrutti Prolog](#)

[Unificazione](#)

[Conjunctive queries](#)

[Le Regole](#)

[Ragionamento](#)

[Liste](#)

[Applicazioni](#)

[Programmazione
nondeterministica](#)

[Unicità di Prolog](#)

programmi = insiemi di assiomi

computazioni = dimostrazioni costruttive di una formula logica data – detta *query* – mediante gli assiomi del programma

Esempio

■ Programma:

- ◆ Socrate è un uomo. (assioma 1)
- ◆ Tutti gli uomini sono mortali. (assioma 2)



L'essenza del paradigma logico

[Paradigma logico](#)

[Prolog](#)

[Costrutti Prolog](#)

[Unificazione](#)

[Conjunctive queries](#)

[Le Regole](#)

[Ragionamento](#)

[Liste](#)

[Applicazioni](#)

[Programmazione
nondeterministica](#)

[Unicità di Prolog](#)

programmi = insiemi di assiomi

computazioni = dimostrazioni costruttive di una formula logica data – detta *query* – mediante gli assiomi del programma

Esempio

■ Programma:

- ◆ Socrate è un uomo. (assioma 1)
- ◆ Tutti gli uomini sono mortali. (assioma 2)

■ Query 1

- ◆ Socrate è un uomo?



L'essenza del paradigma logico

[Paradigma logico](#)

[Prolog](#)

[Costrutti Prolog](#)

[Unificazione](#)

[Conjunctive queries](#)

[Le Regole](#)

[Ragionamento](#)

[Liste](#)

[Applicazioni](#)

[Programmazione
nondeterministica](#)

[Unicità di Prolog](#)

programmi = insiemi di assiomi

computazioni = dimostrazioni costruttive di una formula logica data – detta *query* – mediante gli assiomi del programma

Esempio

■ Programma:

- ◆ Socrate è un uomo. (assioma 1)
- ◆ Tutti gli uomini sono mortali. (assioma 2)

■ Query 1

- ◆ Socrate è un uomo?
- ◆ risposta: *true*



L'essenza del paradigma logico

[Paradigma logico](#)

[Prolog](#)

[Costrutti Prolog](#)

[Unificazione](#)

[Conjunctive queries](#)

[Le Regole](#)

[Ragionamento](#)

[Liste](#)

[Applicazioni](#)

[Programmazione
nondeterministica](#)

[Unicità di Prolog](#)

programmi = insiemi di assiomi

computazioni = dimostrazioni costruttive di una formula logica data – detta *query* – mediante gli assiomi del programma

Esempio

■ Programma:

- ◆ Socrate è un uomo. (assioma 1)
- ◆ Tutti gli uomini sono mortali. (assioma 2)

■ Query 1

- ◆ Socrate è un uomo?
- ◆ risposta: *true*

■ Query 2

- ◆ Socrate è mortale?



L'essenza del paradigma logico

[Paradigma logico](#)

[Prolog](#)

[Costrutti Prolog](#)

[Unificazione](#)

[Conjunctive queries](#)

[Le Regole](#)

[Ragionamento](#)

[Liste](#)

[Applicazioni](#)

[Programmazione
nondeterministica](#)

[Unicità di Prolog](#)

programmi = insiemi di assiomi

computazioni = dimostrazioni costruttive di una formula logica data – detta *query* – mediante gli assiomi del programma

Esempio

■ Programma:

- ◆ Socrate è un uomo. (assioma 1)
- ◆ Tutti gli uomini sono mortali. (assioma 2)

■ Query 1

- ◆ Socrate è un uomo?
- ◆ risposta: *true*

■ Query 2

- ◆ Socrate è mortale?
- ◆ risposta: *true*



L'essenza del paradigma logico

[Paradigma logico](#)

[Prolog](#)

[Costrutti Prolog](#)

[Unificazione](#)

[Conjunctive queries](#)

[Le Regole](#)

[Ragionamento](#)

[Liste](#)

[Applicazioni](#)

[Programmazione
nondeterministica](#)

[Unicità di Prolog](#)

Esempio in Prolog

■ Programma:

- ◆ umano(socrate). (fatto)
- ◆ mortale(X) :- umano(X). (regola)



L'essenza del paradigma logico

[Paradigma logico](#)

[Prolog](#)

[Costrutti Prolog](#)

[Unificazione](#)

[Conjunctive queries](#)

[Le Regole](#)

[Ragionamento](#)

[Liste](#)

[Applicazioni](#)

[Programmazione
nondeterministica](#)

[Unicità di Prolog](#)

Esempio in Prolog

■ Programma:

- ◆ umano(socrate). (fatto)
- ◆ mortale(X) :- umano(X). (regola)

■ Query 1

- ◆ umano(socrate).
- ◆ risposta: *true*



L'essenza del paradigma logico

[Paradigma logico](#)

[Prolog](#)

[Costrutti Prolog](#)

[Unificazione](#)

[Conjunctive queries](#)

[Le Regole](#)

[Ragionamento](#)

[Liste](#)

[Applicazioni](#)

[Programmazione
nondeterministica](#)

[Unicità di Prolog](#)

Esempio in Prolog

■ Programma:

- ◆ umano(socrate). (fatto)
- ◆ mortale(X) :- umano(X). (regola)

■ Query 1

- ◆ umano(socrate).
- ◆ risposta: *true*

■ Query 2

- ◆ mortale(socrate).
- ◆ risposta: *true*



Prolog

[Paradigma logico](#)

Prolog

[Implementazione](#)

[Costrutti Prolog](#)

[Unificazione](#)

[Conjunctive queries](#)

[Le Regole](#)

[Ragionamento](#)

[Liste](#)

[Applicazioni](#)

[Programmazione nondeterministica](#)

[Unicità di Prolog](#)

- Illustreremo assiomi e query mediante il linguaggio di programmazione logica *Prolog*
- Basandoci sul libro *The Art of Prolog* di Sterling e Shapiro, seconda edizione
 - ◆ reperibile in biblioteca e on-line



Implementazione e Interazione in Prolog

[Paradigma logico](#)

[Prolog](#)

Implementazione

[Costrutti Prolog](#)

[Unificazione](#)

[Conjunctive queries](#)

[Le Regole](#)

[Ragionamento](#)

[Liste](#)

[Applicazioni](#)

[Programmazione nondeterministica](#)

[Unicità di Prolog](#)

Analoghe a quelle di ML:

- L'implementazione è mista:



Implementazione e Interazione in Prolog

[Paradigma logico](#)

[Prolog](#)

Implementazione

[Costrutti Prolog](#)

[Unificazione](#)

[Conjunctive queries](#)

[Le Regole](#)

[Ragionamento](#)

[Liste](#)

[Applicazioni](#)

[Programmazione nondeterministica](#)

[Unicità di Prolog](#)

Analoghe a quelle di ML:

■ L'implementazione è mista:

- ◆ I programmi Prolog vengono compilati in un bytecode
- ◆ che viene interpretato da una macchina virtuale
- ◆ la *Warren abstract machine* (WAM)



Implementazione e Interazione in Prolog

[Paradigma logico](#)

[Prolog](#)

Implementazione

[Costrutti Prolog](#)

[Unificazione](#)

[Conjunctive queries](#)

[Le Regole](#)

[Ragionamento](#)

[Liste](#)

[Applicazioni](#)

[Programmazione nondeterministica](#)

[Unicità di Prolog](#)

Analoghe a quelle di ML:

- L'implementazione è mista:
 - ◆ I programmi Prolog vengono compilati in un bytecode
 - ◆ che viene interpretato da una macchina virtuale
 - ◆ la *Warren abstract machine* (WAM)
 - ◆ si possono anche compilare i programmi in codice stand-alone, direttamente eseguibile
- L'interazione con Prolog è analoga a quella con ML
 - ◆ si inviano query all'interprete e si ottengono le relative risposte



Implementazione e Interazione in Prolog

[Paradigma logico](#)

[Prolog](#)

Implementazione

[Costrutti Prolog](#)

[Unificazione](#)

[Conjunctive queries](#)

[Le Regole](#)

[Ragionamento](#)

[Liste](#)

[Applicazioni](#)

[Programmazione nondeterministica](#)

[Unicità di Prolog](#)

Analoghe a quelle di ML:

- L'implementazione è mista:
 - ◆ I programmi Prolog vengono compilati in un bytecode
 - ◆ che viene interpretato da una macchina virtuale
 - ◆ la *Warren abstract machine* (WAM)
 - ◆ si possono anche compilare i programmi in codice stand-alone, direttamente eseguibile
- L'interazione con Prolog è analoga a quella con ML
 - ◆ si inviano query all'interprete e si ottengono le relative risposte
 - ◆ oppure, se il programma è stand alone, si interagisce mediante la sua UI (user interface)



Sistema consigliato per il corso

■ SWI Prolog (free)

- ◆ implementa il Prolog standard
- ◆ supporta sia interpretazione che compilazione stand-alone

[Paradigma logico](#)

[Prolog](#)

[**Implementazione**](#)

[Costrutti Prolog](#)

[Unificazione](#)

[Conjunctive queries](#)

[Le Regole](#)

[Ragionamento](#)

[Liste](#)

[Applicazioni](#)

[Programmazione nondeterministica](#)

[Unicità di Prolog](#)



Sistema consigliato per il corso

Paradigma logico

Prolog

Implementazione

Costrutti Prolog

Unificazione

Conjunctive queries

Le Regole

Ragionamento

Liste

Applicazioni

Programmazione nondeterministica

Unicità di Prolog

■ SWI Prolog (free)

- ◆ implementa il Prolog standard
- ◆ supporta sia interpretazione che compilazione stand-alone
- ◆ invocazione da command line:

```
$ swipl  
Welcome to SWI-Prolog (threaded, 64 bits, version 7.6.4)  
?-
```

(?- è il prompt dell'interprete)



Sistema consigliato per il corso

Paradigma logico

Prolog

Implementazione

Costrutti Prolog

Unificazione

Conjunctive queries

Le Regole

Ragionamento

Liste

Applicazioni

Programmazione nondeterministica

Unicità di Prolog

■ SWI Prolog (free)

- ◆ implementa il Prolog standard
- ◆ supporta sia interpretazione che compilazione stand-alone
- ◆ invocazione da command line:

```
$ swipl  
Welcome to SWI-Prolog (threaded, 64 bits, version 7.6.4)  
?-
```

(?- è il prompt dell'interprete)

■ per caricare un proprio programma mioprog.pl

?- 'mioprog.pl' .	oppure
?- consult('mioprog.pl') .	quando si carica la prima volta;
?- reconsult('mioprog.pl') .	quando si ricarica dopo una correzione



Costrutti base

■ Tre tipi di statement:

1. fatti (facts)
2. regole (rules)
3. queries (detti anche goals)

[Paradigma logico](#)

[Prolog](#)

[**Costrutti Prolog**](#)

Fatti

Queries

Variabili logiche

I termini

I termini

[Unificazione](#)

[Conjunctive queries](#)

[Le Regole](#)

[Ragionamento](#)

[Liste](#)

[Applicazioni](#)

[Programmazione nondeterministica](#)

[Unicità di Prolog](#)



Costrutti base

■ Tre tipi di statement:

1. fatti (facts)
2. regole (rules)
3. queries (detti anche goals)

■ Una sola struttura dati:

1. termini logici (logical terms)

[Paradigma logico](#)

[Prolog](#)

[Costrutti Prolog](#)

Fatti

Queries

Variabili logiche

I termini

I termini

[Unificazione](#)

[Conjunctive queries](#)

[Le Regole](#)

[Ragionamento](#)

[Liste](#)

[Applicazioni](#)

[Programmazione nondeterministica](#)

[Unicità di Prolog](#)



Fatti

- Asseriscono una relazione tra oggetti

```
father(abraham, isaac). (Abramo è il padre di Isacco)
```

father, oltre che *relazione*, è chiamato anche *predicato*

[Paradigma logico](#)

[Prolog](#)

[Costrutti Prolog](#)

Fatti

Queries

Variabili logiche

I termini

I termini

[Unificazione](#)

[Conjunctive queries](#)

[Le Regole](#)

[Ragionamento](#)

[Liste](#)

[Applicazioni](#)

[Programmazione nondeterministica](#)

[Unicità di Prolog](#)



Fatti

Paradigma logico

Prolog

Costrutti Prolog

Fatti

Queries

Variabili logiche

I termini

I termini

Unificazione

Conjunctive queries

Le Regole

Ragionamento

Liste

Applicazioni

Programmazione nondeterministica

Unicità di Prolog

- Asseriscono una relazione tra oggetti

```
father(abraham, isaac). (Abramo è il padre di Isacco)
```

father, oltre che *relazione*, è chiamato anche *predicato*

- Altro esempio: le tabelline

```
per(2,1,2). (due per uno due)  
per(2,2,4). (due per due quattro)  
per(2,3,6). (due per tre sei)  
...
```

- I nomi dei predicati devono iniziare per lettera minuscola



Fatti

Paradigma logico

Prolog

Costrutti Prolog

Fatti

Queries

Variabili logiche

I termini

I termini

Unificazione

Conjunctive queries

Le Regole

Ragionamento

Liste

Applicazioni

Programmazione
nondeterministica

Unicità di Prolog

- Asseriscono una relazione tra oggetti

```
father(abraham, isaac). (Abramo è il padre di Isacco)
```

father, oltre che *relazione*, è chiamato anche *predicato*

- Altro esempio: le tabelline

```
per(2,1,2). (due per uno due)  
per(2,2,4). (due per due quattro)  
per(2,3,6). (due per tre sei)  
...
```

- I nomi dei predicati devono iniziare per lettera minuscola
- Gli argomenti abraham e isaac iniziano con lettera minuscola perchè sono *costanti*, come i numeri
 - ◆ introdurremo le *variabili* più avanti



Fatti

- Con i fatti possiamo definire un *database*
 - ◆ l'esempio più semplice di *programma logico*

father(terach, abraham).	male(terach).
father(terach, nachor).	male(abraham).
father(terach, haran).	male(nachor).
father(abraham, isaac).	male(haran).
father(haran, lot).	male(isaac).
father(haran, milcah).	male(lot).
father(haran, yiscah).	
mother(sarah, isaac).	female(sarah).
	female(milcah).
	female(yiscah).

Program 1.1 A biblical family database

Ogni predicato corrisponde a una *tabella relazionale*



Queries

- I programmi logici sono fatti per rispondere a *queries*
- Se il programma caricato è quello nella slide precedente allora:

```
?- father(abraham,isaac).  
true.
```

```
?- father(isaac,abraham).  
false.
```

Paradigma logico

Prolog

Costrutti Prolog

Fatti

Queries

Variabili logiche

I termini

I termini

Unificazione

Conjunctive queries

Le Regole

Ragionamento

Liste

Applicazioni

Programmazione nondeterministica

Unicità di Prolog



Queries

Paradigma logico

Prolog

Costrutti Prolog

Fatti

Queries

Variabili logiche

I termini

I termini

Unificazione

Conjunctive queries

Le Regole

Ragionamento

Liste

Applicazioni

Programmazione nondeterministica

Unicità di Prolog

- I programmi logici sono fatti per rispondere a *queries*
- Se il programma caricato è quello nella slide precedente allora:

```
?- father(abraham,isaac).  
true.
```

```
?- father(isaac,abraham).  
false.
```

- Le query hanno la stessa forma dei fatti. Sono entrambi dei cosiddetti *atomi logici*.
 - ◆ nei programmi sono asserzioni (fatti)
 - ◆ nelle query sono domande



Queries

Paradigma logico

Prolog

Costrutti Prolog

Fatti

Queries

Variabili logiche

I termini

I termini

Unificazione

Conjunctive queries

Le Regole

Ragionamento

Liste

Applicazioni

Programmazione nondeterministica

Unicità di Prolog

- I programmi logici sono fatti per rispondere a *queries*
- Se il programma caricato è quello nella slide precedente allora:

```
?- father(abraham,isaac).  
true.
```

```
?- father(isaac,abraham).  
false.
```

- Le query hanno la stessa forma dei fatti. Sono entrambi dei cosiddetti *atomi logici*.
 - ◆ nei programmi sono asserzioni (fatti)
 - ◆ nelle query sono domande
- Nell'esempio qui sopra, l'interprete trova il primo atomo nel programma e non trova il secondo (perciò le risposte diverse)



Queries

Paradigma logico

Prolog

Costrutti Prolog

Fatti

Queries

Variabili logiche

I termini

I termini

Unificazione

Conjunctive queries

Le Regole

Ragionamento

Liste

Applicazioni

Programmazione nondeterministica

Unicità di Prolog

- I programmi logici sono fatti per rispondere a *queries*
- Se il programma caricato è quello nella slide precedente allora:

```
?- father(abraham,isaac).  
true.
```

```
?- father(isaac,abraham).  
false.
```

- Le query hanno la stessa forma dei fatti. Sono entrambi dei cosiddetti *atomi logici*.
 - ◆ nei programmi sono asserzioni (fatti)
 - ◆ nelle query sono domande
- Nell'esempio qui sopra, l'interprete trova il primo atomo nel programma e non trova il secondo (perciò le risposte diverse)
- Nota: nel libro le query sono indicate con un punto interrogativo dopo l'atomo, come in `father(abraham,isaac)?`



Variabili logiche nelle query

Paradigma logico

Prolog

Costrutti Prolog

Fatti

Queries

Variabili logiche

I termini

I termini

Unificazione

Conjunctive queries

Le Regole

Ragionamento

Liste

Applicazioni

Programmazione nondeterministica

Unicità di Prolog

- È utile chiedere cose come: *quali sono i figli di abraham?*
- Si può fare con le *variabili logiche*
 - ◆ che si riconoscono perchè iniziano con una *lettera maiuscola*

```
?- father(abraham,X).      /* esiste X tale che father(abraham,X) ? */  
X=isaac.
```



Variabili logiche nelle query

Paradigma logico

Prolog

Costrutti Prolog

Fatti

Queries

Variabili logiche

I termini

I termini

Unificazione

Conjunctive queries

Le Regole

Ragionamento

Liste

Applicazioni

Programmazione
nondeterministica

Unicità di Prolog

- È utile chiedere cose come: *quali sono i figli di abraham?*
- Si può fare con le *variabili logiche*
 - ◆ che si riconoscono perchè iniziano con una *lettera maiuscola*

```
?- father(abraham,X).      /* esiste X tale che father(abraham,X) ? */  
X=isaac.  
  
?- father(terach,X).        /* esiste X tale che father(terach,X) ? */  
X=abraham
```



Variabili logiche nelle query

Paradigma logico

Prolog

Costrutti Prolog

Fatti

Queries

Variabili logiche

I termini

I termini

Unificazione

Conjunctive queries

Le Regole

Ragionamento

Liste

Applicazioni

Programmazione
nondeterministica

Unicità di Prolog

- È utile chiedere cose come: *quali sono i figli di abraham?*
- Si può fare con le *variabili logiche*
 - ◆ che si riconoscono perchè iniziano con una *lettera maiuscola*

```
?- father(abraham,X).      /* esiste X tale che father(abraham,X) ? */  
X=isaac.
```

```
?- father(terach,X).      /* esiste X tale che father(terach,X) ? */  
X=abraham;  
X=nachor
```



Variabili logiche nelle query

Paradigma logico

Prolog

Costrutti Prolog

Fatti

Queries

Variabili logiche

I termini

I termini

Unificazione

Conjunctive queries

Le Regole

Ragionamento

Liste

Applicazioni

Programmazione
nondeterministica

Unicità di Prolog

- È utile chiedere cose come: *quali sono i figli di abraham?*
- Si può fare con le *variabili logiche*
 - ◆ che si riconoscono perchè iniziano con una *lettera maiuscola*

```
?- father(abraham,X).      /* esiste X tale che father(abraham,X) ? */  
X=isaac.  
  
?- father(terach,X).        /* esiste X tale che father(terach,X) ? */  
X=abraham;  
X=nachor;  
X=haran.
```

- La macchina virtuale cerca i valori che – sostituiti a X – rendono la query uguale a uno dei fatti nel programma
 - ◆ NB: questo vale per i programmi di soli fatti...



Variabili logiche in generale

- Differenza tra le variabili logiche e le variabili degli altri paradigmi
 - ◆ Le variabili logiche rappresentano *oggetti qualsiasi*, non specificati
 - ◆ Le variabili dei linguaggi imperativi sono locazioni di memoria
 - ◆ Gli identificatori dei linguaggi funzionali denotano valori immutabili

[Paradigma logico](#)

[Prolog](#)

[Costrutti Prolog](#)

Fatti

Queries

Variabili logiche

I termini

I termini

[Unificazione](#)

[Conjunctive queries](#)

[Le Regole](#)

[Ragionamento](#)

[Liste](#)

[Applicazioni](#)

[Programmazione nondeterministica](#)

[Unicità di Prolog](#)



Variabili logiche in generale

Paradigma logico

Prolog

Costrutti Prolog

Fatti

Queries

Variabili logiche

I termini

I termini

Unificazione

Conjunctive queries

Le Regole

Ragionamento

Liste

Applicazioni

Programmazione
nondeterministica

Unicità di Prolog

■ Differenza tra le variabili logiche e le variabili degli altri paradigmi

- ◆ Le variabili logiche rappresentano *oggetti qualsiasi*, non specificati
- ◆ Le variabili dei linguaggi imperativi sono locazioni di memoria
- ◆ Gli identificatori dei linguaggi funzionali denotano valori immutabili

■ Si possono usare anche nei fatti. Un credente potrebbe scrivere in un programma:

```
creato_da(X,dio). /* ogni cosa è creata da Dio */
```

■ Differenza tra le variabili nei fatti e nelle query:

- ◆ nelle query: *esistenzialmente quantificate*
 - *esiste* un X tale che ... ?
- ◆ nei fatti: *universalmente quantificate*
 - *per ogni* X vale che ...



I termini

- I termini sono l'unica struttura dati in Prolog
- Si costruiscono con costanti, variabili (logiche) e *funtori*
 - ◆ ruolo simile ai costruttori di ML
 - ◆ caratterizzati da nome e *arietà* (il numero di argomenti)

```
<term> ::= <constant>
          | <variable>
          | <functor name> '()' [<term> [ ',' <term> ] * ] ''
```

[Paradigma logico](#)

[Prolog](#)

[Costrutti Prolog](#)

Fatti

Queries

Variabili logiche

I termini

I termini

[Unificazione](#)

[Conjunctive queries](#)

[Le Regole](#)

[Ragionamento](#)

Liste

[Applicazioni](#)

[Programmazione nondeterministica](#)

[Unicità di Prolog](#)



I termini

- I termini sono l'unica struttura dati in Prolog
- Si costruiscono con costanti, variabili (logiche) e *funtori*
 - ◆ ruolo simile ai costruttori di ML
 - ◆ caratterizzati da nome e *arietà* (il numero di argomenti)

```
<term> ::= <constant>
          | <variable>
          | <functor name> '()' [<term> [ ',' <term>]*] '()
```

Esempi:

- Costanti: a, pippo, 10, 3.14, "ciao"
- Variabili: X, Y, Persona
- Termini composti:

```
successor(Numero)
date(25, aprile, 2020)
color(rgb, 0, 0, 1)
```

- Non ci sono dichiarazioni di tipo
- Costanti simboliche e funtori si usano senza dichiararli prima.



I termini

- I termini sono l'unica struttura dati in Prolog
- Si costruiscono con costanti, variabili (logiche) e *funtori*
 - ◆ ruolo simile ai costruttori di ML
 - ◆ caratterizzati da nome e *arietà* (il numero di argomenti)

```
<term> ::= <constant>
          | <variable>
          | <functor name> '()' [<term> [ ',' <term>]*] '()
```

[Paradigma logico](#)

[Prolog](#)

[Costrutti Prolog](#)

Fatti

Queries

Variabili logiche

I termini

I termini

[Unificazione](#)

[Conjunctive queries](#)

[Le Regole](#)

[Ragionamento](#)

[Liste](#)

[Applicazioni](#)

[Programmazione nondeterministica](#)

[Unicità di Prolog](#)



I termini

- I termini sono l'unica struttura dati in Prolog
- Si costruiscono con costanti, variabili (logiche) e *funtori*
 - ◆ ruolo simile ai costruttori di ML
 - ◆ caratterizzati da nome e *arietà* (il numero di argomenti)

```
<term> ::= <constant>
          | <variable>
          | <functor name> '()' [<term> [ ',' <term>]*] '()
```

Esempi

```
successor(Int)
date(25,april,2020)
color(rgb,0,0,1)
```

- Non ci sono dichiarazioni di tipo
- Costanti simboliche e funtori si usano senza dichiararli prima.



I termini

- Gli argomenti di un predicato possono essere termini qualsiasi

```
born(john , date(10,october,2000)).  
hasColor(object1 , color(rgb,1,0,0)).
```

[Paradigma logico](#)

[Prolog](#)

[Costrutti Prolog](#)

Fatti

Queries

Variabili logiche

I termini

I termini

[Unificazione](#)

[Conjunctive queries](#)

[Le Regole](#)

[Ragionamento](#)

[Liste](#)

[Applicazioni](#)

[Programmazione nondeterministica](#)

[Unicità di Prolog](#)



I termini

Paradigma logico

Prolog

Costrutti Prolog

Fatti

Queries

Variabili logiche

I termini

I termini

Unificazione

Conjunctive queries

Le Regole

Ragionamento

Liste

Applicazioni

Programmazione
nondeterministica

Unicità di Prolog

- Gli argomenti di un predicato possono essere termini qualsiasi

```
born(john , date(10,october,2000)).  
hasColor(object1 , color(rgb,1,0,0)).
```

- Quindi la sintassi generale dei fatti è

```
<fact> ::= <atomic formula> '.'  
  
<atomic formula> ::=  
    <predicate name> '(' [<term> [ ',' <term>]*] ')'
```

(notare il punto dopo il fatto)



I termini

Paradigma logico

Prolog

Costrutti Prolog

Fatti

Queries

Variabili logiche

I termini

I termini

Unificazione

Conjunctive queries

Le Regole

Ragionamento

Liste

Applicazioni

Programmazione
nondeterministica

Unicità di Prolog

- Gli argomenti di un predicato possono essere termini qualsiasi

```
born(john , date(10,october,2000)).  
hasColor(object1 , color(rgb,1,0,0)).
```

- Quindi la sintassi generale dei fatti è

```
<fact> ::= <atomic formula> '.'  
  
<atomic formula> ::=  
    <predicate name> '(' [<term> [ ',' <term> ] * ] ')'
```

(notare il punto dopo il fatto)

- Esempi di query a un programma coi fatti qui sopra

```
?- born(X, date(Y,october,2000)).  
X=john ,  
Y=10 .
```



I termini

Paradigma logico

Prolog

Costrutti Prolog

Fatti

Queries

Variabili logiche

I termini

I termini

Unificazione

Conjunctive queries

Le Regole

Ragionamento

Liste

Applicazioni

Programmazione
nondeterministica

Unicità di Prolog

- Gli argomenti di un predicato possono essere termini qualsiasi

```
born(john , date(10,october,2000)).  
hasColor(object1 , color(rgb,1,0,0)).
```

- Quindi la sintassi generale dei fatti è

```
<fact> ::= <atomic formula> '.'  
  
<atomic formula> ::=  
    <predicate name> '(' [<term> [ ',' <term> ] * ] ')'
```

(notare il punto dopo il fatto)

- Esempi di query a un programma coi fatti qui sopra

```
?- born(X, date(Y,october,2000)).  
    X=john ,  
    Y=10 .
```

```
?- hasColor(object1 , Y).  
    Y=color(rgb,1,0,0).
```



I termini

- La stessa variabile X può comparire più volte nello stesso fatto o nella stessa query
- Significa che i termini nelle posizioni dove si trova X devono essere uguali tra loro

[Paradigma logico](#)

[Prolog](#)

[Costrutti Prolog](#)

Fatti

Queries

Variabili logiche

I termini

I termini

[Unificazione](#)

[Conjunctive queries](#)

[Le Regole](#)

[Ragionamento](#)

[Liste](#)

[Applicazioni](#)

[Programmazione nondeterministica](#)

[Unicità di Prolog](#)



I termini

- La stessa variabile X può comparire più volte nello stesso fatto o nella stessa query
- Significa che i termini nelle posizioni dove si trova X devono essere uguali tra loro

```
/* fatti */  
div(X,X,1).  
sum(X,0,X).  
sum(0,X,X).
```

[Paradigma logico](#)

[Prolog](#)

[Costrutti Prolog](#)

Fatti

Queries

Variabili logiche

I termini

I termini

[Unificazione](#)

[Conjunctive queries](#)

[Le Regole](#)

[Ragionamento](#)

[Liste](#)

[Applicazioni](#)

[Programmazione nondeterministica](#)

[Unicità di Prolog](#)



I termini

- La stessa variabile X può comparire più volte nello stesso fatto o nella stessa query
- Significa che i termini nelle posizioni dove si trova X devono essere uguali tra loro

```
/* fatti */  
div(X,X,1).  
sum(X,0,X).  
sum(0,X,X).  
  
/* query */  
?- hasColor(object1, color(rgb,Y,Y,Y)).  
    false.                                /* nell'unico fatto su object1 c'è 1,0,0 */
```

[Paradigma logico](#)

[Prolog](#)

[Costrutti Prolog](#)

Fatti

Queries

Variabili logiche

I termini

I termini

[Unificazione](#)

[Conjunctive queries](#)

[Le Regole](#)

[Ragionamento](#)

[Liste](#)

[Applicazioni](#)

[Programmazione nondeterministica](#)

[Unicità di Prolog](#)



I termini

- La stessa variabile X può comparire più volte nello stesso fatto o nella stessa query
- Significa che i termini nelle posizioni dove si trova X devono essere uguali tra loro

```
/* fatti */  
div(X,X,1).  
sum(X,0,X).  
sum(0,X,X).  
  
/* query */  
?- hasColor(object1, color(rgb,Y,Y,Y)).  
    false.                      /* nell'unico fatto su object1 c'è 1,0,0 */
```

- Differenza tra termini di Prolog e pattern di ML
 - ◆ in ML non posso usare la stessa variabile più volte nello stesso pattern
 - ◆ perchè servono solo a estrarre informazioni da una struttura, non a controllare se elementi diversi sono uguali



Ground e nonground

- Un termine è *ground* se non contiene variabili
- Altrimenti è *nonground*

```
foo(a,b)      /* ground      */  
bar(X)        /* nonground */
```

[Paradigma logico](#)

[Prolog](#)

[Costrutti Prolog](#)

Fatti

Queries

Variabili logiche

I termini

I termini

[Unificazione](#)

[Conjunctive queries](#)

[Le Regole](#)

[Ragionamento](#)

[Liste](#)

[Applicazioni](#)

[Programmazione nondeterministica](#)

[Unicità di Prolog](#)



Ground e nonground

- Un termine è *ground* se non contiene variabili
- Altrimenti è *nonground*

```
foo(a,b)      /* ground      */  
bar(X)        /* nonground */
```

- Gli stessi aggettivi e gli stessi criteri si applicano ai fatti, alle query e anche alle regole (che vedremo più avanti)

```
father(abraham,isaac).    /* ground      */  
sum(X,0,X).                /* nonground */
```

[Paradigma logico](#)

[Prolog](#)

[Costrutti Prolog](#)

Fatti

Queries

Variabili logiche

I termini

I termini

Unificazione

[Conjunctive queries](#)

[Le Regole](#)

Ragionamento

Liste

Applicazioni

Programmazione
nondeterministica

Unicità di Prolog



Paradigma logico

Prolog

Costrutti Prolog

Unificazione

Sostituzioni

Istanze

L'unificazione

L'unificazione

Search tree

Conjunctive queries

Conjunctive queries

Le Regole

Ragionamento

Liste

Applicazioni

Programmazione
nondeterministica

Unicità di Prolog

Unificazione



Come si risponde alle query

[Paradigma logico](#)

[Prolog](#)

[Costrutti Prolog](#)

[Unificazione](#)

Sostituzioni

Istanze

L'unificazione

L'unificazione

Search tree

Conjunctive queries

[Conjunctive queries](#)

[Le Regole](#)

[Ragionamento](#)

[Liste](#)

[Applicazioni](#)

Programmazione
nondeterministica

[Unicità di Prolog](#)

- Per spiegarlo dobbiamo specificare come avviene il matching tra query e fatti e come si costruiscono le risposte
- Dovremo introdurre due nozioni:
 - ◆ sostituzione
 - ◆ unificazione



Sostituzioni

■ **Definizione:** Una *sostituzione* è un insieme finito di coppie

$$\theta = \{X_1 = t_1, \dots X_n = t_n\}$$

dove

- ◆ Le X_i sono variabili e i t_i termini.
- ◆ Le X_i sono tutte diverse tra loro.
- ◆ Nessuna delle X_i compare dentro i t_i .

Paradigma logico

Prolog

Costrutti Prolog

Unificazione

Sostituzioni

Istanze

L'unificazione

L'unificazione

Search tree

Conjunctive queries

Conjunctive queries

Le Regole

Ragionamento

Liste

Applicazioni

Programmazione
nondeterministica

Unicità di Prolog



Sostituzioni

Paradigma logico

Prolog

Costrutti Prolog

Unificazione

Sostituzioni

Istanze

L'unificazione

L'unificazione

Search tree

Conjunctive queries

Conjunctive queries

Le Regole

Ragionamento

Liste

Applicazioni

Programmazione
nondeterministica

Unicità di Prolog

- **Definizione:** Una *sostituzione* è un insieme finito di coppie

$$\theta = \{X_1 = t_1, \dots X_n = t_n\}$$

dove

- ◆ Le X_i sono variabili e i t_i termini.
- ◆ Le X_i sono tutte diverse tra loro.
- ◆ Nessuna delle X_i compare dentro i t_i .

- L'*applicazione* di θ a una espressione E (che potrebbe essere un termine, un fatto, una query o una regola)

- ◆ si denota con $E\theta$
- ◆ sostituisce le occorrenze delle variabili X_i in E con i rispettivi termini t_i



Sostituzioni

- **Definizione:** Una *sostituzione* è un insieme finito di coppie

$$\theta = \{X_1 = t_1, \dots X_n = t_n\}$$

dove

- ◆ Le X_i sono variabili e i t_i termini.
 - ◆ Le X_i sono tutte diverse tra loro.
 - ◆ Nessuna delle X_i compare dentro i t_i .
- L'applicazione di θ a una espressione E (che potrebbe essere un termine, un fatto, una query o una regola)
 - ◆ si denota con $E\theta$
 - ◆ sostituisce le occorrenze delle variabili X_i in E con i rispettivi termini t_i
 - Esempio: se $\theta = \{X = \text{isaac}\}$ e $E = \text{father}(\text{abraham}, X)$ allora

$$E\theta = \text{father}(\text{abraham}, \text{isaac})$$



Istanze

- L'applicazione di una sostituzione θ a E crea un “caso particolare” di E ,
 - ◆ dove le variabili di E (che indicano oggetti non specificati)
 - ◆ vengono sostituite con valori specifici (termini ground)
 - ◆ o parzialmente specificati (termini nonground)

[Paradigma logico](#)

[Prolog](#)

[Costrutti Prolog](#)

[Unificazione](#)

[Sostituzioni](#)

Istanze

L'unificazione

L'unificazione

Search tree

Conjunctive queries

Conjunctive queries

[Le Regole](#)

[Ragionamento](#)

[Liste](#)

[Applicazioni](#)

[Programmazione nondeterministica](#)

[Unicità di Prolog](#)



Istanze

- L'applicazione di una sostituzione θ a E crea un "caso particolare" di E ,
 - ◆ dove le variabili di E (che indicano oggetti non specificati)
 - ◆ vengono sostituite con valori specifici (termini ground)
 - ◆ o parzialmente specificati (termini nonground)
- Esempio: $\theta = \{X = \text{color(rgb, Y, Y, Y)}\}$ e $E = \text{hasColor(o1, X)}$.

[Paradigma logico](#)

[Prolog](#)

[Costrutti Prolog](#)

[Unificazione](#)

[Sostituzioni](#)

Istanze

L'unificazione

L'unificazione

Search tree

Conjunctive queries

Conjunctive queries

[Le Regole](#)

[Ragionamento](#)

[Liste](#)

[Applicazioni](#)

[Programmazione nondeterministica](#)

[Unicità di Prolog](#)



Istanze

- L'applicazione di una sostituzione θ a E crea un "caso particolare" di E ,
 - ◆ dove le variabili di E (che indicano oggetti non specificati)
 - ◆ vengono sostituite con valori specifici (termini ground)
 - ◆ o parzialmente specificati (termini nonground)
- Esempio: $\theta = \{X = \text{color(rgb, Y, Y, Y)}\}$ e $E = \text{hasColor(o1, X)}$.
 - ◆ $E\theta = \text{hasColor(o1, color(rgb, Y, Y, Y))}$

[Paradigma logico](#)

[Prolog](#)

[Costrutti Prolog](#)

[Unificazione](#)

[Sostituzioni](#)

Istanze

[L'unificazione](#)

[L'unificazione](#)

[Search tree](#)

[Conjunctive queries](#)

[Conjunctive queries](#)

[Le Regole](#)

[Ragionamento](#)

[Liste](#)

[Applicazioni](#)

[Programmazione nondeterministica](#)

[Unicità di Prolog](#)



Istanze

- L'applicazione di una sostituzione θ a E crea un "caso particolare" di E ,
 - ◆ dove le variabili di E (che indicano oggetti non specificati)
 - ◆ vengono sostituite con valori specifici (termini ground)
 - ◆ o parzialmente specificati (termini nonground)
- Esempio: $\theta = \{X = \text{color(rgb, Y, Y, Y)}\}$ e $E = \text{hasColor(o1, X)}$.
 - ◆ $E\theta = \text{hasColor(o1, color(rgb, Y, Y, Y))}$
 - ◆ E dice che o1 ha un colore non specificato

[Paradigma logico](#)

[Prolog](#)

[Costrutti Prolog](#)

[Unificazione](#)

[Sostituzioni](#)

Istanze

[L'unificazione](#)

[L'unificazione](#)

[Search tree](#)

[Conjunctive queries](#)

[Conjunctive queries](#)

[Le Regole](#)

[Ragionamento](#)

[Liste](#)

[Applicazioni](#)

[Programmazione nondeterministica](#)

[Unicità di Prolog](#)



Istanze

- L'applicazione di una sostituzione θ a E crea un “caso particolare” di E ,
 - ◆ dove le variabili di E (che indicano oggetti non specificati)
 - ◆ vengono sostituite con valori specifici (termini ground)
 - ◆ o parzialmente specificati (termini nonground)
- Esempio: $\theta = \{X = \text{color(rgb, Y, Y, Y)}\}$ e $E = \text{hasColor(o1, X)}$.
 - ◆ $E\theta = \text{hasColor(o1, color(rgb, Y, Y, Y))}$
 - ◆ E dice che o1 ha un colore non specificato
 - ◆ $E\theta$ lo specifica parzialmente: è in formato rgb e tutti i 3 valori sono uguali

[Paradigma logico](#)

[Prolog](#)

[Costrutti Prolog](#)

[Unificazione](#)

[Sostituzioni](#)

Istanze

[L'unificazione](#)

[L'unificazione](#)

[Search tree](#)

[Conjunctive queries](#)

Conjunctive queries

[Le Regole](#)

[Ragionamento](#)

[Liste](#)

[Applicazioni](#)

[Programmazione nondeterministica](#)

[Unicità di Prolog](#)



Istanze

Paradigma logico

Prolog

Costrutti Prolog

Unificazione

Sostituzioni

Istanze

L'unificazione

L'unificazione

Search tree

Conjunctive queries

Conjunctive queries

Le Regole

Ragionamento

Liste

Applicazioni

Programmazione nondeterministica

Unicità di Prolog

- L'applicazione di una sostituzione θ a E crea un “caso particolare” di E ,
 - ◆ dove le variabili di E (che indicano oggetti non specificati)
 - ◆ vengono sostituite con valori specifici (termini ground)
 - ◆ o parzialmente specificati (termini nonground)
- Esempio: $\theta = \{X = \text{color(rgb, Y, Y, Y)}\}$ e $E = \text{hasColor(o1, X)}$.
 - ◆ $E\theta = \text{hasColor(o1, color(rgb, Y, Y, Y))}$
 - ◆ E dice che o1 ha un colore non specificato
 - ◆ $E\theta$ lo specifica parzialmente: è in formato rgb e tutti i 3 valori sono uguali
- Definizione: E_1 è un'istanza di E_2 se esiste una sostituzione θ tale che $E_1 = E_2\theta$
 - ◆ cioè se E_1 è un “caso particolare” di E_2 dove alcune variabili di E_2 sono istanziate, cioè legate a un valore



L'unificazione

- L'algoritmo di unificazione è quello che effettua il matching
- Prende due espressioni E_1 ed E_2 e – se possibile – restituisce una sostituzione θ tale che

$$E_1\theta = E_2\theta$$

detta *unificatore* (unifier)

Paradigma logico

Prolog

Costrutti Prolog

Unificazione

Sostituzioni

Istanze

L'unificazione

L'unificazione

Search tree

Conjunctive queries

Conjunctive queries

Le Regole

Ragionamento

Liste

Applicazioni

Programmazione
nondeterministica

Unicità di Prolog



L'unificazione

Paradigma logico

Prolog

Costrutti Prolog

Unificazione

Sostituzioni

Istanze

L'unificazione

L'unificazione

Search tree

Conjunctive queries

Conjunctive queries

Le Regole

Ragionamento

Liste

Applicazioni

Programmazione
nondeterministica

Unicità di Prolog

- L'algoritmo di unificazione è quello che effettua il matching
- Prende due espressioni E_1 ed E_2 e – se possibile – restituisce una sostituzione θ tale che

$$E_1\theta = E_2\theta$$

detta *unificatore* (unifier)

- L'unificatore costruito dall'algoritmo viene detto *most general unifier* (mgu) perchè
 - ◆ non sostituisce una variabile con un termine se non è necessario, cioè
 - ◆ vincola il meno possibile il risultato $E_1\theta$, lasciando le variabili libere quando può



L'unificazione

Paradigma logico

Prolog

Costrutti Prolog

Unificazione

Sostituzioni

Istanze

L'unificazione

L'unificazione

Search tree

Conjunctive queries

Conjunctive queries

Le Regole

Ragionamento

Liste

Applicazioni

Programmazione nondeterministica

Unicità di Prolog

- L'algoritmo di unificazione è quello che effettua il matching
- Prende due espressioni E_1 ed E_2 e – se possibile – restituisce una sostituzione θ tale che

$$E_1\theta = E_2\theta$$

detta *unificatore* (unifier)

- L'unificatore costruito dall'algoritmo viene detto *most general unifier* (mgu) perchè
 - ◆ non sostituisce una variabile con un termine se non è necessario, cioè
 - ◆ vincola il meno possibile il risultato $E_1\theta$, lasciando le variabili libere quando può
 - ◆ tecnicamente, ogni altro unificatore θ' porta a una *istanza* (cioè caso particolare) di $E_1\theta$, cioè
 - ◆ esiste una sostituzione σ tale che $(E_1\theta)\sigma = E_1\theta'$



L'unificazione

[Paradigma logico](#)

[Prolog](#)

[Costrutti Prolog](#)

[Unificazione](#)

Sostituzioni

Istanze

L'unificazione

L'unificazione

Search tree

Conjunctive queries

Conjunctive queries

Le Regole

Ragionamento

Liste

Applicazioni

Programmazione
nondeterministica

Unicità di Prolog

- **Definizione:** Date due espressioni E_1, E_2 , un *unificatore* è una sostituzione θ tale che $E_1\theta = E_2\theta$.
- **Definizione:** Date due espressioni E_1, E_2 , un *unificatore più generale* (*most general unifier, mgu*) è un unificatore θ tale che per ogni altro unificatore θ' esiste una sostituzione σ tale che $E_1\theta' = E_1\theta\sigma$.



L'unificazione

[Paradigma logico](#)

[Prolog](#)

[Costrutti Prolog](#)

[Unificazione](#)

Sostituzioni

Istanze

L'unificazione

L'unificazione

Search tree

Conjunctive queries

[Conjunctive queries](#)

[Le Regole](#)

Ragionamento

Liste

Applicazioni

Programmazione
nondeterministica

Unicità di Prolog

Esempi

- ◆ Le seguenti sostituzioni sono tutte unificatori delle espressioni $E_1 = \text{sum}(A, B, C)$ ed $E_2 = \text{sum}(X, 0, X)$:
 - $\theta_0 = \{A = 0, B = 0, C = 0, X = 0\}$
 - $\theta_1 = \{A = 1, B = 0, C = 1, X = 1\}$
 - etc.
 - $\sigma = \{A = X, B = 0, C = X\}$
- ◆ ...ma una sola è la *most general* (mgu)
- ◆ $\text{mgu}(\text{sum}(A, B, C), \text{sum}(X, 0, X)) = \{A = X, B = 0, C = X\}$



L'unificazione

[Paradigma logico](#)

[Prolog](#)

[Costrutti Prolog](#)

[Unificazione](#)

Sostituzioni

Istanze

L'unificazione

L'unificazione

Search tree

Conjunctive queries

[Conjunctive queries](#)

Le Regole

Ragionamento

Liste

Applicazioni

Programmazione
nondeterministica

Unicità di Prolog

Esempi

- ◆ Le seguenti sostituzioni sono tutte unificatori delle espressioni $E_1 = \text{sum}(A, B, C)$ ed $E_2 = \text{sum}(X, 0, X)$:
 - $\theta_0 = \{A = 0, B = 0, C = 0, X = 0\}$
 - $\theta_1 = \{A = 1, B = 0, C = 1, X = 1\}$
 - etc.
 - $\sigma = \{A = X, B = 0, C = X\}$
- ◆ ...ma una sola è la *most general* (mgu)
 - ◆ $\text{mgu}(\text{sum}(A, B, C), \text{sum}(X, 0, X)) = \{A = X, B = 0, C = X\}$
 - ◆ $\text{mgu}(\text{sum}(0, X, 0), \text{sum}(Y, 0, Y)) = \{X = 0, Y = 0\}$



L'unificazione

Paradigma logico

Prolog

Costrutti Prolog

Unificazione

Sostituzioni

Istanze

L'unificazione

L'unificazione

Search tree

Conjunctive queries

Conjunctive queries

Le Regole

Ragionamento

Liste

Applicazioni

Programmazione
nondeterministica

Unicità di Prolog

Esempi

- ◆ Le seguenti sostituzioni sono tutte unificatori delle espressioni $E_1 = \text{sum}(A, B, C)$ ed $E_2 = \text{sum}(X, 0, X)$:
 - $\theta_0 = \{A = 0, B = 0, C = 0, X = 0\}$
 - $\theta_1 = \{A = 1, B = 0, C = 1, X = 1\}$
 - etc.
 - $\sigma = \{A = X, B = 0, C = X\}$
- ◆ ...ma una sola è la *most general* (mgu)
 - ◆ $\text{mgu}(\text{sum}(A, B, C), \text{sum}(X, 0, X)) = \{A = X, B = 0, C = X\}$
 - ◆ $\text{mgu}(\text{sum}(0, X, 0), \text{sum}(Y, 0, Y)) = \{X = 0, Y = 0\}$
 - ◆ $\text{mgu}(\text{sum}(0, X, 0), \text{sum}(Y, Z, 1))$ non esiste a causa del terzo argomento (non si può rendere 0=1)



Esercizi sull'unificazione

Paradigma logico

Prolog

Costrutti Prolog

Unificazione

Sostituzioni

Istanze

L'unificazione

L'unificazione

Search tree

Conjunctive queries

Conjunctive queries

Le Regole

Ragionamento

Liste

Applicazioni

Programmazione
nondeterministica

Unicità di Prolog

1. Quali delle seguenti sostituzioni sono unificatori per le espressioni
 $E_1 = \text{hasColor}(X, \text{color}(\text{rgb}, \text{Red}, 0, \text{Blue}))$ e
 $E_2 = \text{hasColor}(\text{box}, \text{color}(\text{Format}, 0, Y, Y))$:
 - (a) $\{\text{Red} = 0\}$
 - (b) $\{\text{Red} = Z, Z = 0, \text{Blue} = Y\}$
 - (c) $\{X = \text{box}, \text{Red} = 0, \text{Blue} = Y\}$
 - (d) $\{X = \text{box}, \text{Red} = 0, \text{Blue} = 0, Y = 0\}$
 - (e) $\{X = \text{box}, \text{Red} = 0, \text{Blue} = 1, Y = 1, \text{Format} = \text{rgb}\}$
 - (f) $\{X = \text{box}, \text{Red} = 0, \text{Blue} = 0, Y = 0, \text{Format} = \text{rgb}\}$
2. Elencare almeno 3 unificatori, tra cui l'mgu, delle seguenti espressioni:
 $E_1 = \text{has_type}(a, \text{record}(\text{array}(X), Y, \text{pointer}(Z)))$,
 $E_2 = \text{has_type}(V, \text{record}(T, \text{float}, \text{pointer}(\text{void})))$



Costruzione delle riposte da soli fatti

- Nei programmi visti sinora (che consistono di soli fatti) le risposte a una query

$$q(t_1, \dots, t_n)$$

vengono costruite così

1. si cerca nel programma il primo fatto $p(u_1, \dots, u_m)$ con lo stesso funtore (cioè $p = q$ e $m = n$). Se se ne trova uno:

[Paradigma logico](#)

[Prolog](#)

[Costrutti Prolog](#)

[Unificazione](#)

Sostituzioni

Istanze

L'unificazione

L'unificazione

Search tree

Conjunctive queries

[Conjunctive queries](#)

[Le Regole](#)

[Ragionamento](#)

[Liste](#)

[Applicazioni](#)

Programmazione
nondeterministica

Unicità di Prolog



Costruzione delle riposte da soli fatti

- Nei programmi visti sinora (che consistono di soli fatti) le risposte a una query

$$q(t_1, \dots, t_n)$$

vengono costruite così

1. si cerca nel programma il primo fatto $p(u_1, \dots, u_m)$ con lo stesso funtore (cioè $p = q$ e $m = n$). Se se ne trova uno:
2. si calcola $\theta = \text{mgu}(q(t_1, \dots, t_n), p(u_1, \dots, u_m))$
3. se θ esiste, allora:

[Paradigma logico](#)

[Prolog](#)

[Costrutti Prolog](#)

[Unificazione](#)

Sostituzioni

Istanze

L'unificazione

L'unificazione

Search tree

Conjunctive queries

[Conjunctive queries](#)

[Le Regole](#)

[Ragionamento](#)

[Liste](#)

[Applicazioni](#)

Programmazione
nondeterministica

[Unicità di Prolog](#)



Costruzione delle riposte da soli fatti

- Nei programmi visti sinora (che consistono di soli fatti) le risposte a una query

$$q(t_1, \dots, t_n)$$

vengono costruite così

1. si cerca nel programma il primo fatto $p(u_1, \dots, u_m)$ con lo stesso funtore (cioè $p = q$ e $m = n$). Se se ne trova uno:
 - ◆ si calcola $\theta = \text{mgu}(q(t_1, \dots, t_n), p(u_1, \dots, u_m))$
 - 3. se θ esiste, allora:
 - ◆ si restituisce θ come risposta (se è vuota allora Prolog dice *true*)
 - ◆ poi se l'utente non vuole altre soluzioni si termina
4. altrimenti si cerca il prossimo fatto con lo stesso funtore. Se esiste si ripete da 2, altrimenti si termina

[Paradigma logico](#)

[Prolog](#)

[Costrutti Prolog](#)

[Unificazione](#)

Sostituzioni

Istanze

L'unificazione

L'unificazione

Search tree

Conjunctive queries

[Conjunctive queries](#)

[Le Regole](#)

[Ragionamento](#)

[Liste](#)

[Applicazioni](#)

Programmazione
nondeterministica

Unicità di Prolog



Costruzione delle riposte da soli fatti

Paradigma logico

Prolog

Costrutti Prolog

Unificazione

Sostituzioni

Istanze

L'unificazione

L'unificazione

Search tree

Conjunctive queries

Conjunctive queries

Le Regole

Ragionamento

Liste

Applicazioni

Programmazione
nondeterministica

Unicità di Prolog

- Nei programmi visti sinora (che consistono di soli fatti) le risposte a una query

$$q(t_1, \dots, t_n)$$

vengono costruite così

1. si cerca nel programma il primo fatto $p(u_1, \dots, u_m)$ con lo stesso funtore (cioè $p = q$ e $m = n$). Se se ne trova uno:
 - ◆ si calcola $\theta = \text{mgu}(q(t_1, \dots, t_n), p(u_1, \dots, u_m))$
 - 3. se θ esiste, allora:
 - ◆ si restituisce θ come risposta (se è vuota allora Prolog dice *true*)
 - ◆ poi se l'utente non vuole altre soluzioni si termina
 4. altrimenti si cerca il prossimo fatto con lo stesso funtore. Se esiste si ripete da 2, altrimenti si termina
- Prolog dice *false* quando nessun fatto unifica con la query



Rappresentazione grafica del procedimento

Paradigma logico

Prolog

Costrutti Prolog

Unificazione

Sostituzioni

Istanze

L'unificazione

L'unificazione

Search tree

Conjunctive queries

Conjunctive queries

Le Regole

Ragionamento

Liste

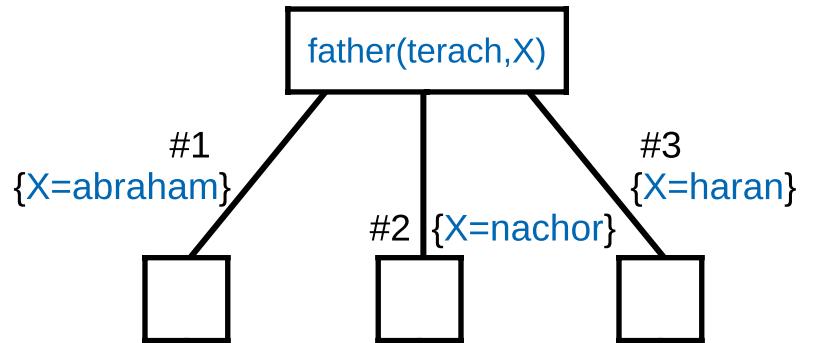
Applicazioni

Programmazione
nondeterministica

Unicità di Prolog

- Un *search tree* per la query `father(terach,X)`.

```
/* programma */  
/*1*/ father(terach, abraham).  
/*2*/ father(terach, nachor).  
/*3*/ father(terach, haran).  
/*4*/ father(abraham, isaac).  
    ...
```





Rappresentazione grafica del procedimento

Paradigma logico

Prolog

Costrutti Prolog

Unificazione

Sostituzioni

Istanze

L'unificazione

L'unificazione

Search tree

Conjunctive queries

Conjunctive queries

Le Regole

Ragionamento

Liste

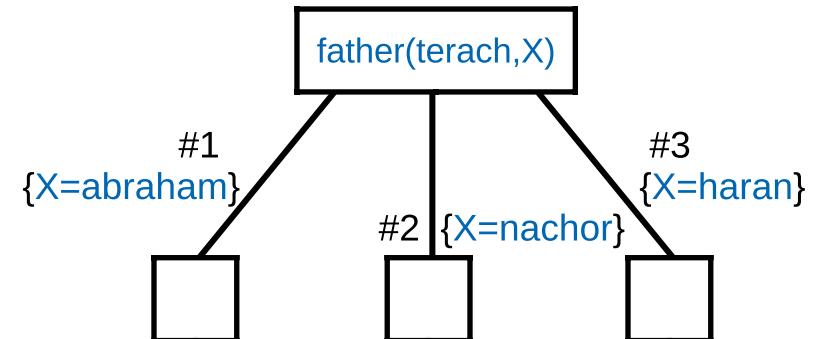
Applicazioni

Programmazione nondeterministica

Unicità di Prolog

- Un *search tree* per la query `father(terach,X)`.

```
/* programma */  
/*1*/ father(terach, abraham).  
/*2*/ father(terach, nachor).  
/*3*/ father(terach, haran).  
/*4*/ father(abraham, isaac).  
    ...
```



- Gli archi corrispondono ai fatti che unificano con la query
- Sono etichettati con
 - ◆ il numero del fatto utilizzato (nell'ordine in cui compare nel programma)
 - ◆ il mgu della query e del fatto



Rappresentazione grafica del procedimento

Paradigma logico

Prolog

Costrutti Prolog

Unificazione

Sostituzioni

Istanze

L'unificazione

L'unificazione

Search tree

Conjunctive queries

Conjunctive queries

Le Regole

Ragionamento

Liste

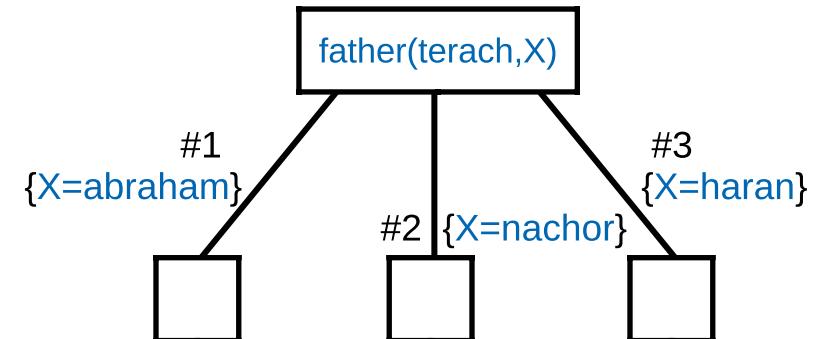
Applicazioni

Programmazione nondeterministica

Unicità di Prolog

- Un *search tree* per la query `father(terach,X)`.

```
/* programma */  
/*1*/ father(terach, abraham).  
/*2*/ father(terach, nachor).  
/*3*/ father(terach, haran).  
/*4*/ father(abraham, isaac).  
    ...
```



- Gli archi corrispondono ai fatti che unificano con la query
- Sono etichettati con
 - ◆ il numero del fatto utilizzato (nell'ordine in cui compare nel programma)
 - ◆ il mgu della query e del fatto
- Il search tree di una query (rispetto a un programma) comprende *tutte* le risposte che Prolog genera se l'utente glielo chiede



Rappresentazione grafica del procedimento

Paradigma logico

Prolog

Costrutti Prolog

Unificazione

Sostituzioni

Istanze

L'unificazione

L'unificazione

Search tree

Conjunctive queries

Conjunctive queries

Le Regole

Ragionamento

Liste

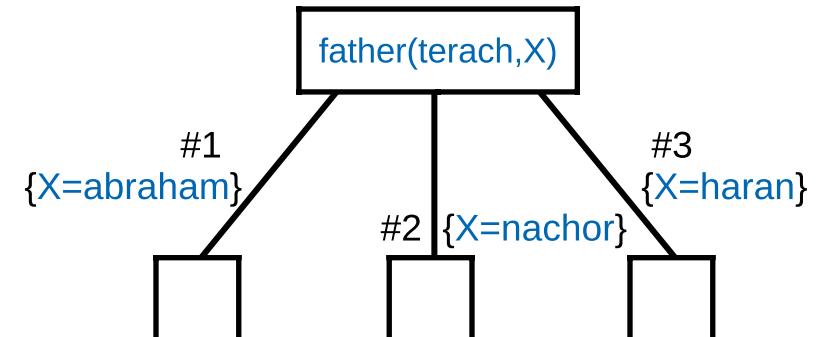
Applicazioni

Programmazione nondeterministica

Unicità di Prolog

- Un *search tree* per la query `father(terach,X)`.

```
/* programma */  
/*1*/ father(terach, abraham).  
/*2*/ father(terach, nachor).  
/*3*/ father(terach, haran).  
/*4*/ father(abraham, isaac).  
    ...
```



- Gli archi corrispondono ai fatti che unificano con la query
- Sono etichettati con
 - ◆ il numero del fatto utilizzato (nell'ordine in cui compare nel programma)
 - ◆ il mgu della query e del fatto
- Il search tree di una query (rispetto a un programma) comprende *tutte* le risposte che Prolog genera se l'utente glielo chiede
- Le computazioni di Prolog corrispondono a una visita depth-first da sinistra a destra (ogni ramo di successo una risposta)



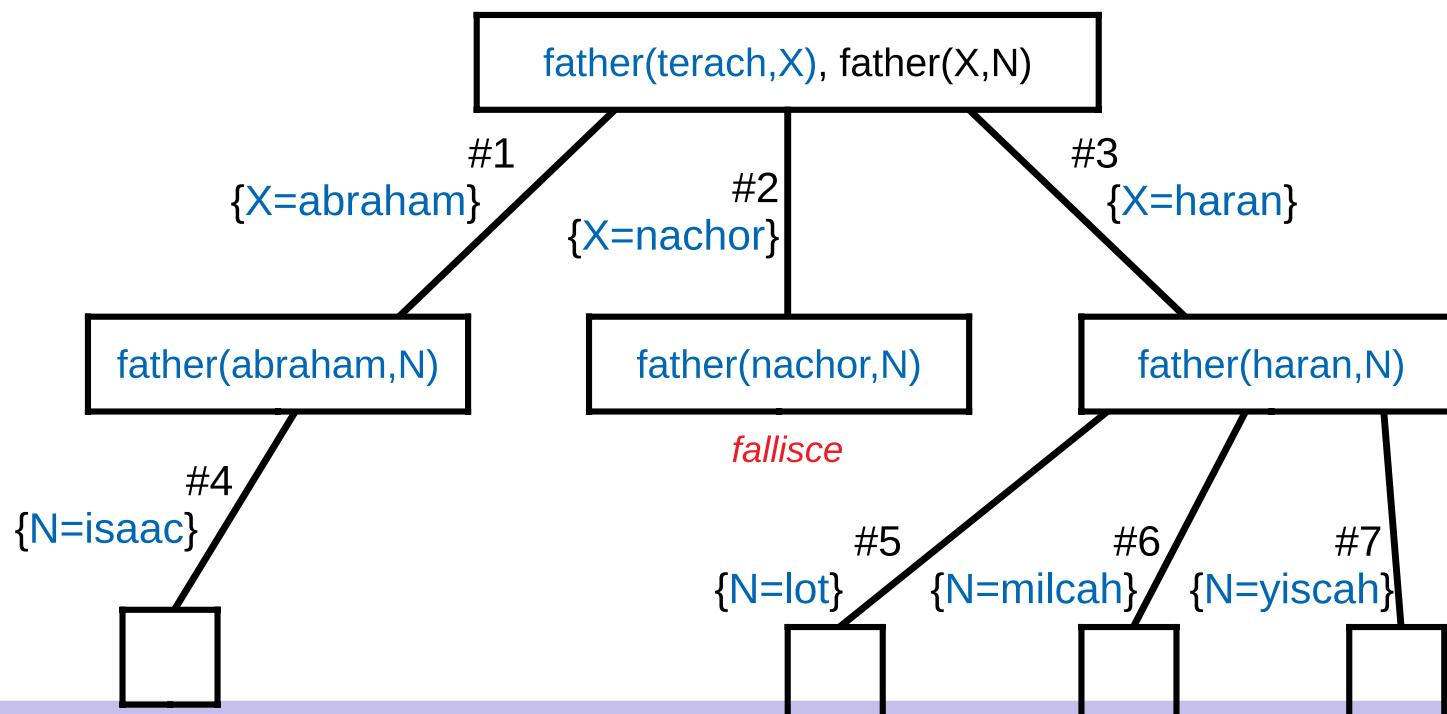
Conjunctive queries

- Le query possono contenere più formule atomiche (*goals*)
- Esempio: chi sono i nipoti di terach?

`father(terach,X), father(X,Nipote)`

(la virgola è un *and*) è come un *join*

```
/* programma */  
/*1*/ father(terach, abraham).  
/*2*/ father(terach, nachor).  
/*3*/ father(terach, haran).  
/*4*/ father(abraham, isaac).  
/*5*/ father(haran, lot).  
/*6*/ father(haran, milcah).  
/*7*/ father(haran, yiscah).  
...
```





Conjunctive queries

- Le figlie di haran le trovo con la query congiuntiva ... ?

[Paradigma logico](#)

[Prolog](#)

[Costrutti Prolog](#)

[Unificazione](#)

Conjunctive queries

[Le Regole](#)

[Ragionamento](#)

[Liste](#)

[Applicazioni](#)

[Programmazione
nondeterministica](#)

[Unicità di Prolog](#)



Conjunctive queries

- Le figlie di haran le trovo con la query congiuntiva ... ?

```
father(haran ,Y) , female(Y).
```

I valori di Y mi danno le figlie di haran

[Paradigma logico](#)

[Prolog](#)

[Costrutti Prolog](#)

[Unificazione](#)

Conjunctive queries

[Le Regole](#)

[Ragionamento](#)

[Liste](#)

[Applicazioni](#)

[Programmazione
nondeterministica](#)

[Unicità di Prolog](#)



Conjunctive queries

Paradigma logico

Prolog

Costrutti Prolog

Unificazione

Conjunctive queries

Le Regole

Ragionamento

Liste

Applicazioni

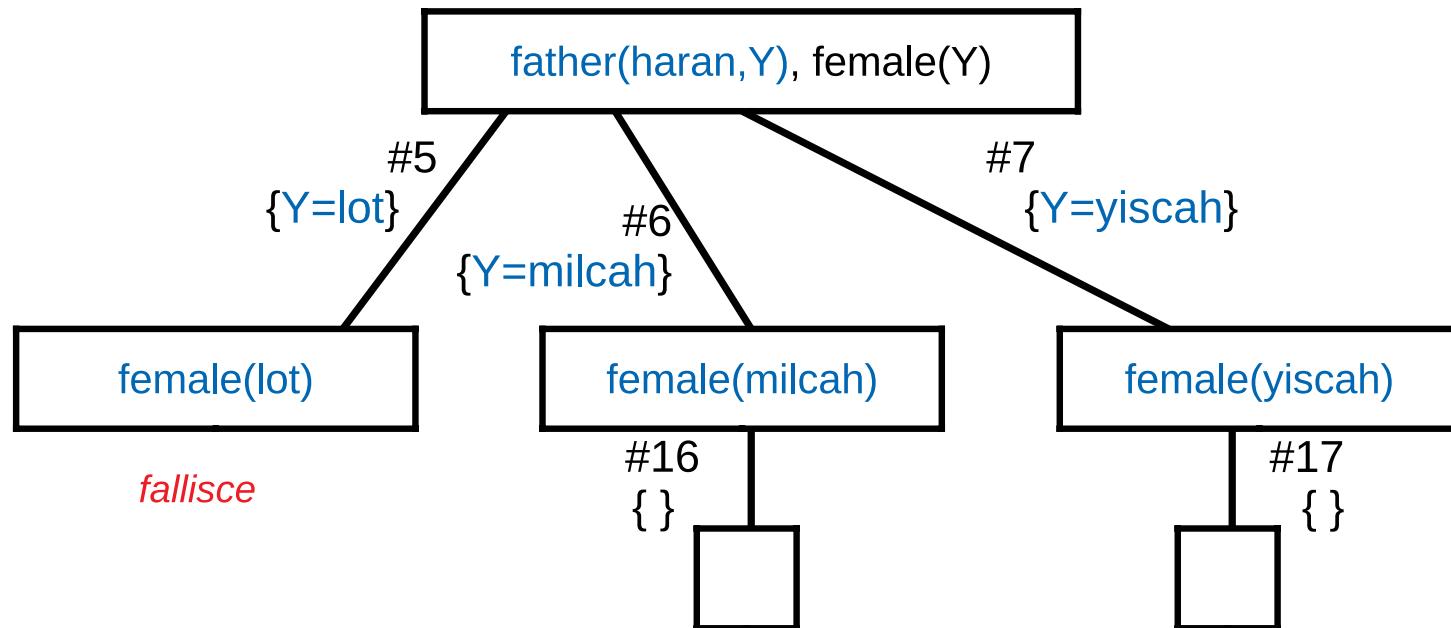
Programmazione
nondeterministica

Unicità di Prolog

- Le figlie di haran le trovo con la query congiuntiva ... ?

```
father(haran, Y), female(Y).
```

I valori di Y mi danno le figlie di haran



```
Y = milcah;
Y = yiscah;
false.
```



Conjunctive queries

Paradigma logico

Prolog

Costrutti Prolog

Unificazione

Conjunctive queries

Le Regole

Ragionamento

Liste

Applicazioni

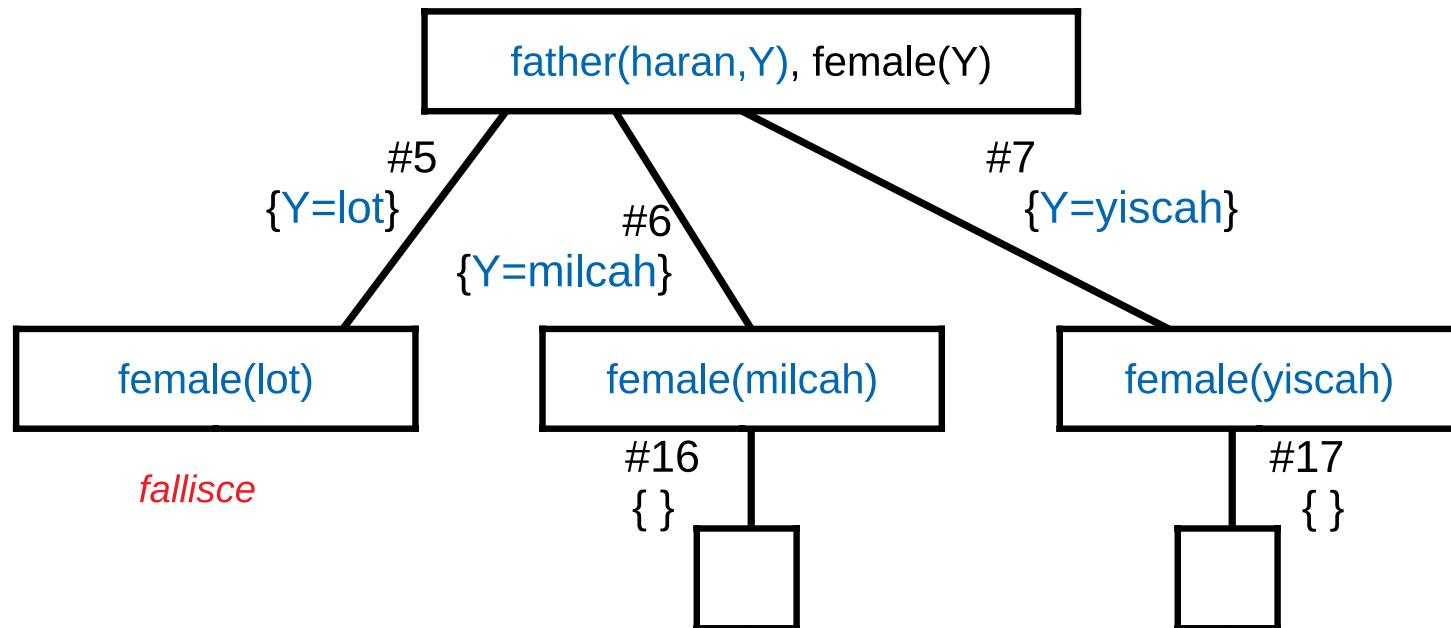
Programmazione nondeterministica

Unicità di Prolog

- Le figlie di haran le trovo con la query congiuntiva ... ?

```
father(haran, Y), female(Y).
```

I valori di Y mi danno le figlie di haran



```
Y = milcah;
Y = yiscah;
false.
```

- Ma... perchè non definire il concetto di figlia nel programma?



[Paradigma logico](#)

[Prolog](#)

[Costrutti Prolog](#)

[Unificazione](#)

[Conjunctive queries](#)

Le Regole

[Ragionamento](#)

[Liste](#)

[Applicazioni](#)

[Programmazione
nondeterministica](#)

[Unicità di Prolog](#)

Le Regole



Le regole

■ *X è figlia di Y se Y è padre di X e X è femmina*

```
daughter(X,Y) :- father(Y,X), female(X).
```

[Paradigma logico](#)

[Prolog](#)

[Costrutti Prolog](#)

[Unificazione](#)

[Conjunctive queries](#)

[Le Regole](#)

[Ragionamento](#)

[Liste](#)

[Applicazioni](#)

[Programmazione
nondeterministica](#)

[Unicità di Prolog](#)



Le regole

[Paradigma logico](#)

[Prolog](#)

[Costrutti Prolog](#)

[Unificazione](#)

[Conjunctive queries](#)

[Le Regole](#)

[Ragionamento](#)

[Liste](#)

[Applicazioni](#)

[Programmazione
nondeterministica](#)

[Unicità di Prolog](#)

- X è figlia di Y se Y è padre di X e X è femmina

```
daughter(X,Y) :- father(Y,X), female(X).
```

- In generale la sintassi delle regole è

```
<rule> ::= <atomic formula> [:- <goal list>].
```

```
<goal list> ::= <atomic formula> [, <atomic formula>]*
```



Le regole

[Paradigma logico](#)

[Prolog](#)

[Costrutti Prolog](#)

[Unificazione](#)

[Conjunctive queries](#)

[Le Regole](#)

[Ragionamento](#)

[Liste](#)

[Applicazioni](#)

[Programmazione
nondeterministica](#)

[Unicità di Prolog](#)

- X è figlia di Y se Y è padre di X e X è femmina

```
daughter(X,Y) :- father(Y,X), female(X).
```

- In generale la sintassi delle regole è

```
<rule> ::= <atomic formula> [:- <goal list>].
```

```
<goal list> ::= <atomic formula> [, <atomic formula>]*
```

- ◆ $:-$ sta per \Leftarrow
- ◆ la formula atomica prima di $:-$ è detta **testa (head)**



Le regole

Paradigma logico

Prolog

Costrutti Prolog

Unificazione

Conjunctive queries

Le Regole

Ragionamento

Liste

Applicazioni

Programmazione
nondeterministica

Unicità di Prolog

- X è figlia di Y se Y è padre di X e X è femmina

```
daughter(X,Y) :- father(Y,X), female(X).
```

- In generale la sintassi delle regole è

```
<rule> ::= <atomic formula> [:- <goal list>].
```

```
<goal list> ::= <atomic formula> [, <atomic formula>]*
```

- ◆ :- sta per \Leftarrow
- ◆ la formula atomica prima di :- è detta **testa (head)**
- ◆ la parte dopo :- è detta **corpo (body)**



Le regole

[Paradigma logico](#)

[Prolog](#)

[Costrutti Prolog](#)

[Unificazione](#)

[Conjunctive queries](#)

[Le Regole](#)

[Ragionamento](#)

[Liste](#)

[Applicazioni](#)

[Programmazione
nondeterministica](#)

[Unicità di Prolog](#)

- X è figlia di Y se Y è padre di X e X è femmina

```
daughter(X,Y) :- father(Y,X), female(X).
```

- In generale la sintassi delle regole è

```
<rule> ::= <atomic formula> [:- <goal list>].
```

```
<goal list> ::= <atomic formula> [, <atomic formula>]*
```

- ◆ :- sta per \Leftarrow
- ◆ la formula atomica prima di :- è detta **testa (head)**
- ◆ la parte dopo :- è detta **corpo (body)**
- ◆ notare che i fatti non sono che regole senza corpo



Le regole

[Paradigma logico](#)

[Prolog](#)

[Costrutti Prolog](#)

[Unificazione](#)

[Conjunctive queries](#)

[Le Regole](#)

[Ragionamento](#)

[Liste](#)

[Applicazioni](#)

[Programmazione
nondeterministica](#)

[Unicità di Prolog](#)

- X è figlia di Y se Y è padre di X e X è femmina

```
daughter(X,Y) :- father(Y,X), female(X).
```

- In generale la sintassi delle regole è

```
<rule> ::= <atomic formula> [:- <goal list>].
```

```
<goal list> ::= <atomic formula> [, <atomic formula>]*
```

- ◆ `:-` sta per \Leftarrow
- ◆ la formula atomica prima di `:-` è detta **testa (head)**
- ◆ la parte dopo `:-` è detta **corpo (body)**
- ◆ notare che i fatti non sono che regole senza corpo

- **Definizione:** un programma logico è una sequenza di regole



[Paradigma logico](#)

[Prolog](#)

[Costrutti Prolog](#)

[Unificazione](#)

[Conjunctive queries](#)

[Le Regole](#)

[**Ragionamento**](#)

Derivazioni

Overloading

Wildcards

Analisi di circuiti

Cammini su grafi

Cammini su grafi

Cammini su grafi

Cammini su grafi

Relational algebra

Negazione

[Liste](#)

[Applicazioni](#)

Programmazione

nondeterministica

[Unicità di Prolog](#)

Ragionamento



Ragionare con le regole

- Aggiungiamo in coda al programma (posizione 18) la regola

```
daughter(X,Y) :- father(Y,X), female(X).
```

Le risposte alla query `daughter(Z, haran)` si trovano così:

[Paradigma logico](#)

[Prolog](#)

[Costrutti Prolog](#)

[Unificazione](#)

[Conjunctive queries](#)

[Le Regole](#)

[Ragionamento](#)

[Derivazioni](#)

Overloading

Wildcards

Analisi di circuiti

Cammini su grafi

Cammini su grafi

Cammini su grafi

Cammini su grafi

Relational algebra

Negazione

[Liste](#)

[Applicazioni](#)

[Programmazione nondeterministica](#)

[Unicità di Prolog](#)



Ragionare con le regole

Paradigma logico

Prolog

Costrutti Prolog

Unificazione

Conjunctive queries

Le Regole

Ragionamento

Derivazioni

Overloading

Wildcards

Analisi di circuiti

Cammini su grafi

Cammini su grafi

Cammini su grafi

Cammini su grafi

Relational algebra

Negazione

Liste

Applicazioni

Programmazione nondeterministica

Unicità di Prolog

- Aggiungiamo in coda al programma (posizione 18) la regola

```
daughter(X,Y) :- father(Y,X), female(X).
```

Le risposte alla query `daughter(Z, haran)` si trovano così:

```
daughter(Z,haran)
```

Cerca una testa che unifica con la query
e sostituisce la query col corpo
istanziato con il mgu



Ragionare con le regole

Paradigma logico

Prolog

Costrutti Prolog

Unificazione

Conjunctive queries

Le Regole

Ragionamento

Derivazioni

Overloading

Wildcards

Analisi di circuiti

Cammini su grafi

Cammini su grafi

Cammini su grafi

Cammini su grafi

Relational algebra

Negazione

Liste

Applicazioni

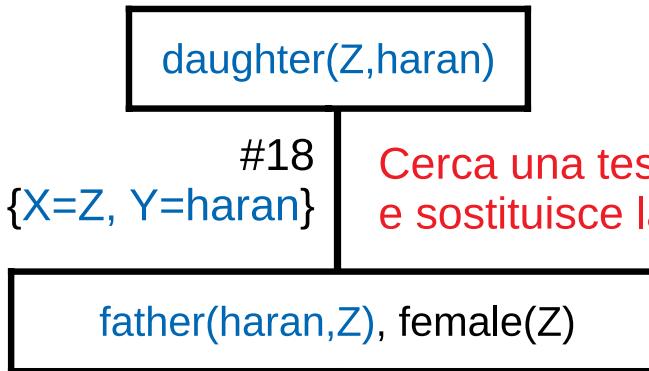
Programmazione nondeterministica

Unicità di Prolog

- Aggiungiamo in coda al programma (posizione 18) la regola

```
daughter(X,Y) :- father(Y,X), female(X).
```

Le risposte alla query `daughter(Z, haran)` si trovano così:



Cerca una testa che unifica con la query
e sostituisce la query col corpo
istanziato con il mgu



Ragionare con le regole

Paradigma logico

Prolog

Costrutti Prolog

Unificazione

Conjunctive queries

Le Regole

Ragionamento

Derivazioni

Overloading

Wildcards

Analisi di circuiti

Cammini su grafi

Cammini su grafi

Cammini su grafi

Cammini su grafi

Relational algebra

Negazione

Liste

Applicazioni

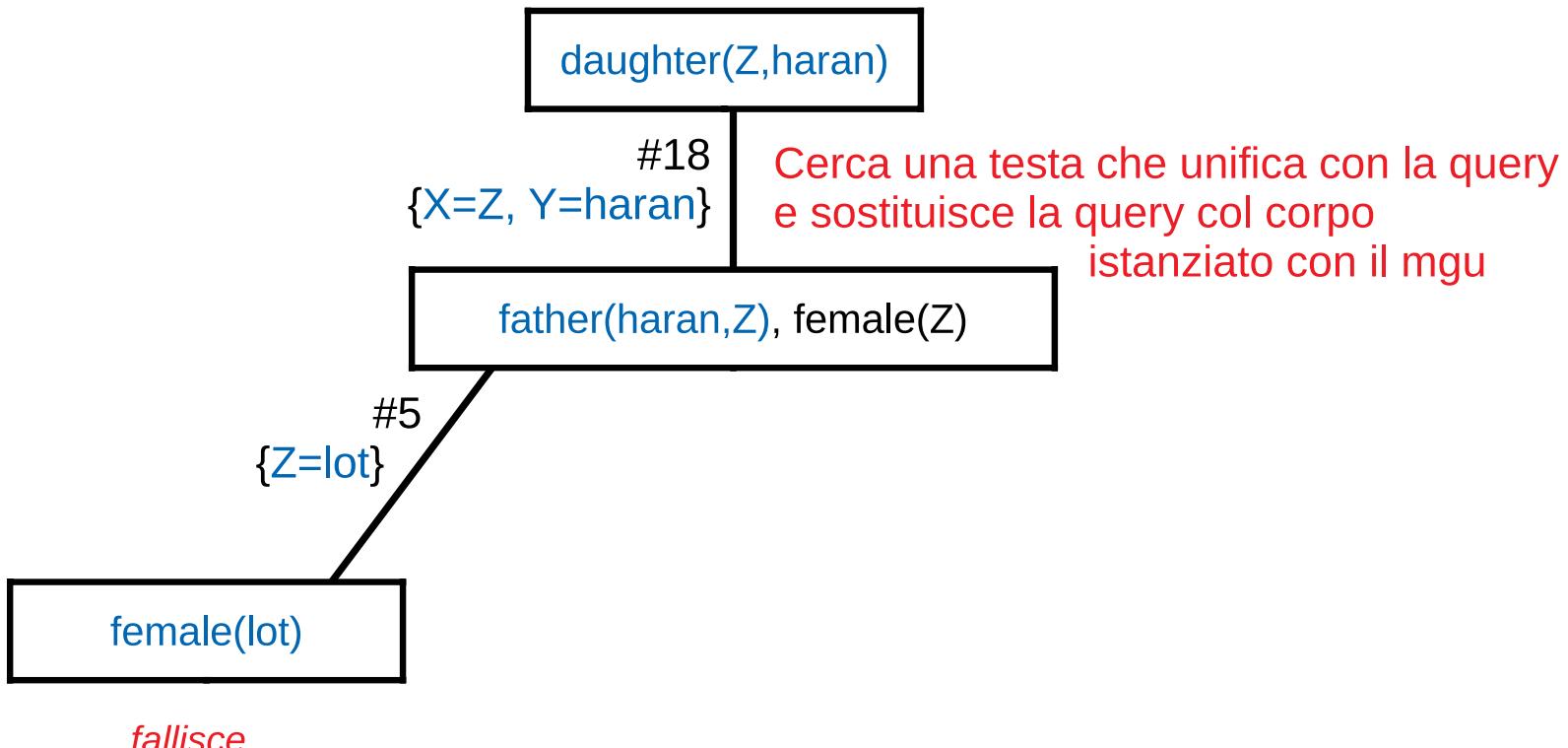
Programmazione
nondeterministica

Unicità di Prolog

- Aggiungiamo in coda al programma (posizione 18) la regola

```
daughter(X,Y) :- father(Y,X), female(X).
```

Le risposte alla query `daughter(Z, haran)` si trovano così:





Ragionare con le regole

Paradigma logico

Prolog

Costrutti Prolog

Unificazione

Conjunctive queries

Le Regole

Ragionamento

Derivazioni

Overloading

Wildcards

Analisi di circuiti

Cammini su grafi

Cammini su grafi

Cammini su grafi

Cammini su grafi

Relational algebra

Negazione

Liste

Applicazioni

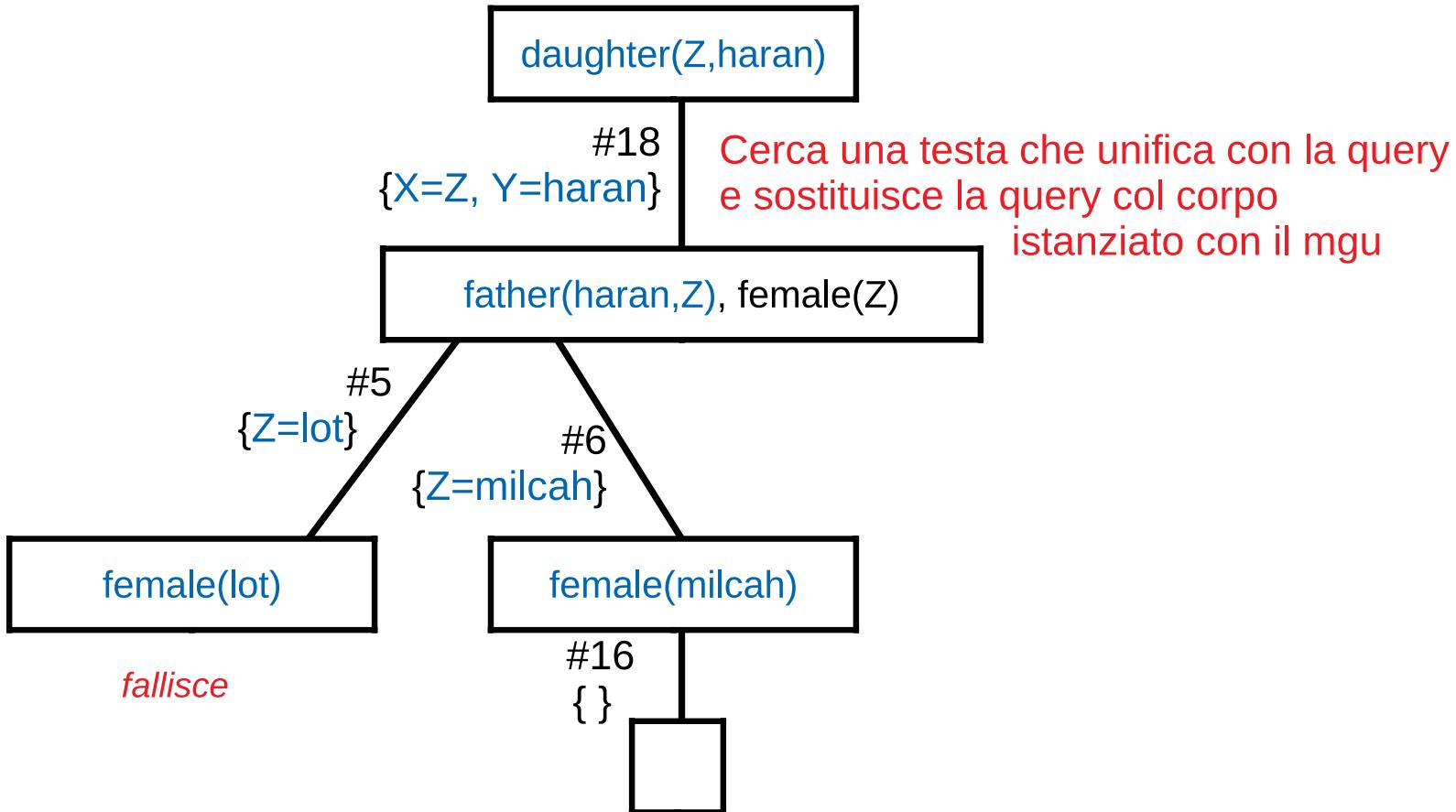
Programmazione
nondeterministica

Unicità di Prolog

- Aggiungiamo in coda al programma (posizione 18) la regola

```
daughter(X,Y) :- father(Y,X), female(X).
```

Le risposte alla query `daughter(Z, haran)` si trovano così:





Ragionare con le regole

Paradigma logico

Prolog

Costrutti Prolog

Unificazione

Conjunctive queries

Le Regole

Ragionamento

Derivazioni

Overloading

Wildcards

Analisi di circuiti

Cammini su grafi

Cammini su grafi

Cammini su grafi

Cammini su grafi

Relational algebra

Negazione

Liste

Applicazioni

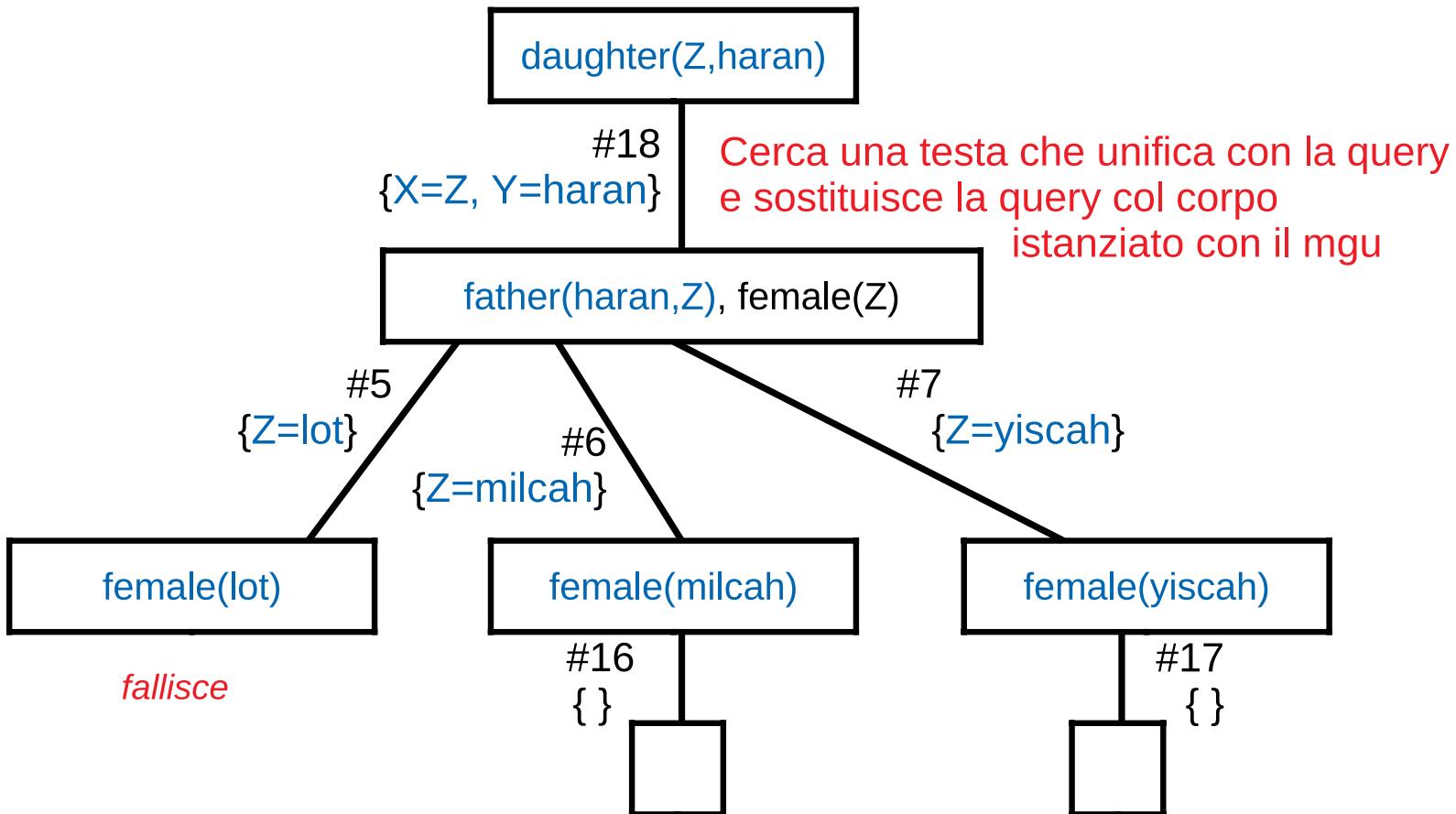
Programmazione nondeterministica

Unicità di Prolog

- Aggiungiamo in coda al programma (posizione 18) la regola

```
daughter(X,Y) :- father(Y,X), female(X).
```

Le risposte alla query `daughter(Z, haran)` si trovano così:





Ragionare con le regole

[Paradigma logico](#)

[Prolog](#)

[Costrutti Prolog](#)

[Unificazione](#)

[Conjunctive queries](#)

[Le Regole](#)

[Ragionamento](#)

[Derivazioni](#)

Overloading

Wildcards

Analisi di circuiti

Cammini su grafi

Cammini su grafi

Cammini su grafi

Cammini su grafi

Relational algebra

Negazione

[Liste](#)

[Applicazioni](#)

[Programmazione
nondeterministica](#)

[Unicità di Prolog](#)

■ Bisogna fare attenzione con le variabili

◆ quelle della query devono sempre essere diverse da quelle della regola



Ragionare con le regole

Paradigma logico

Prolog

Costrutti Prolog

Unificazione

Conjunctive queries

Le Regole

Ragionamento

Derivazioni

Overloading

Wildcards

Analisi di circuiti

Cammini su grafi

Cammini su grafi

Cammini su grafi

Cammini su grafi

Relational algebra

Negazione

Liste

Applicazioni

Programmazione
nondeterministica

Unicità di Prolog

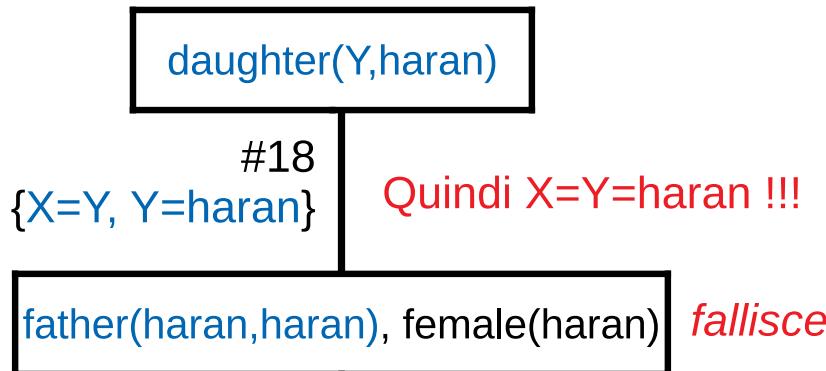
- Bisogna fare attenzione con le variabili

- ◆ quelle della query devono sempre essere diverse da quelle della regola

Esempio di problema se non fosse così:

```
daughter(X,Y) :- father(Y,X), female(X).
```

Le risposte alla query `daughter(Y, haran)` verrebbero perse





Ragionare con le regole

Paradigma logico

Prolog

Costrutti Prolog

Unificazione

Conjunctive queries

Le Regole

Ragionamento

Derivazioni

Overloading

Wildcards

Analisi di circuiti

Cammini su grafi

Cammini su grafi

Cammini su grafi

Cammini su grafi

Relational algebra

Negazione

Liste

Applicazioni

Programmazione
nondeterministica

Unicità di Prolog

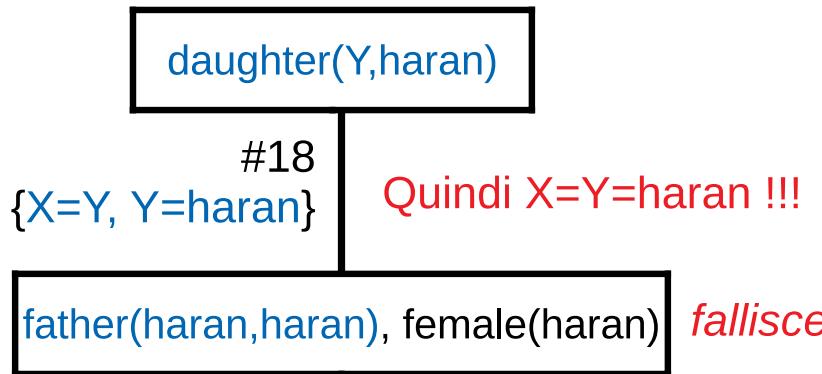
- Bisogna fare attenzione con le variabili

- ◆ quelle della query devono sempre essere diverse da quelle della regola

Esempio di problema se non fosse così:

```
daughter(X,Y) :- father(Y,X), female(X).
```

Le risposte alla query `daughter(Y, haran)` verrebbero perse



- Per evitare questo problema, ogni volta che una regola viene usata la WAM ridenomina le sue variabili, ad esempio

```
daughter(_101,_102) :- father(_102,_101), female(_101).
```



Migliorando l'esempio

- Con la definizione attuale ci perdiamo le figlie delle donne
- **Soluzione 1:** aggiungere un'altra regola in fondo al programma

```
daughter(X, Y) :- mother(Y, X), female(X).
```

che mi permette di dedurre che X è figlia di Y quando Y è madre di X e X è femmina

Paradigma logico

Prolog

Costrutti Prolog

Unificazione

Conjunctive queries

Le Regole

Ragionamento

Derivazioni

Overloading

Wildcards

Analisi di circuiti

Cammini su grafi

Cammini su grafi

Cammini su grafi

Cammini su grafi

Relational algebra

Negazione

Liste

Applicazioni

Programmazione
nondeterministica

Unicità di Prolog



Migliorando l'esempio

Paradigma logico

Prolog

Costrutti Prolog

Unificazione

Conjunctive queries

Le Regole

Ragionamento

Derivazioni

Overloading

Wildcards

Analisi di circuiti

Cammini su grafi

Cammini su grafi

Cammini su grafi

Cammini su grafi

Relational algebra

Negazione

Liste

Applicazioni

Programmazione nondeterministica

Unicità di Prolog

- Con la definizione attuale ci perdiamo le figlie delle donne
- **Soluzione 1:** aggiungere un'altra regola in fondo al programma

```
daughter(X, Y) :- mother(Y, X), female(X).
```

che mi permette di dedurre che X è figlia di Y quando Y è madre di X e X è femmina

- **Soluzione 2:** introdurre il concetto di *genitore* (*parent*)

```
parent(X, Y) :- father(X, Y).  
parent(X, Y) :- mother(X, Y).
```



Migliorando l'esempio

Paradigma logico

Prolog

Costrutti Prolog

Unificazione

Conjunctive queries

Le Regole

Ragionamento

Derivazioni

Overloading

Wildcards

Analisi di circuiti

Cammini su grafi

Cammini su grafi

Cammini su grafi

Cammini su grafi

Relational algebra

Negazione

Liste

Applicazioni

Programmazione
nondeterministica

Unicità di Prolog

- Con la definizione attuale ci perdiamo le figlie delle donne
- **Soluzione 1:** aggiungere un'altra regola in fondo al programma

```
daughter(X, Y) :- mother(Y, X), female(X).
```

che mi permette di dedurre che X è figlia di Y quando Y è madre di X e X è femmina

- **Soluzione 2:** introdurre il concetto di *genitore* (parent)

```
parent(X, Y) :- father(X, Y).  
parent(X, Y) :- mother(X, Y).
```

/ puo' essere riutilizzato per diverse definizioni */*

```
daughter(X, Y) :- parent(Y, X), female(X).
```

```
son(X, Y) :- parent(Y, X), male(X).
```

```
grandparent(X, Y) :- parent(X, Z), parent(Z, Y).
```



Esempio di derivazione

[Paradigma logico](#)

[Prolog](#)

[Costrutti Prolog](#)

[Unificazione](#)

[Conjunctive queries](#)

[Le Regole](#)

[Ragionamento](#)

Derivazioni

Overloading

Wildcards

Analisi di circuiti

Cammini su grafi

Cammini su grafi

Cammini su grafi

Cammini su grafi

Relational algebra

Negazione

[Liste](#)

[Applicazioni](#)

[Programmazione nondeterministica](#)

[Unicità di Prolog](#)

```
/* 4 */ father(abraham,isaac).      ....  
/* 8 */ mother(sarah,isaac).       ....  
/*13*/ male(isaac).             ....  
/*19*/ parent(X,Y) :- father(X,Y).  
/*20*/ parent(X,Y) :- mother(X,Y).  
/*21*/ son(X,Y)      :- parent(Y,X), male(X).
```



Esempio di derivazione

[Paradigma logico](#)

[Prolog](#)

[Costrutti Prolog](#)

[Unificazione](#)

[Conjunctive queries](#)

[Le Regole](#)

[Ragionamento](#)

[Derivazioni](#)

Overloading

Wildcards

Analisi di circuiti

Cammini su grafi

Cammini su grafi

Cammini su grafi

Cammini su grafi

Relational algebra

Negazione

[Liste](#)

[Applicazioni](#)

Programmazione
nondeterministica

[Unicità di Prolog](#)

```
/* 4 */ father(abraham,isaac).      ....  
/* 8 */ mother(sarah,isaac).       ....  
/*13*/ male(isaac).             ....  
/*19*/ parent(X,Y) :- father(X,Y).  
/*20*/ parent(X,Y) :- mother(X,Y).  
/*21*/ son(X,Y)      :- parent(Y,X), male(X).
```

Query:

son(isaac,Z)



Esempio di derivazione

[Paradigma logico](#)

[Prolog](#)

[Costrutti Prolog](#)

[Unificazione](#)

[Conjunctive queries](#)

[Le Regole](#)

[Ragionamento](#)

[Derivazioni](#)

Overloading

Wildcards

Analisi di circuiti

Cammini su grafi

Cammini su grafi

Cammini su grafi

Cammini su grafi

Relational algebra

Negazione

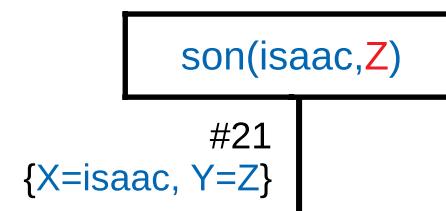
[Liste](#)

[Applicazioni](#)

[Programmazione nondeterministica](#)

[Unicità di Prolog](#)

```
/* 4 */ father(abraham,isaac).      ....  
/* 8 */ mother(sarah,isaac).       ....  
/*13*/ male(isaac).             ....  
/*19*/ parent(X,Y) :- father(X,Y).  
/*20*/ parent(X,Y) :- mother(X,Y).  
/*21*/ son(X,Y)      :- parent(Y,X), male(X).
```





Esempio di derivazione

[Paradigma logico](#)

[Prolog](#)

[Costrutti Prolog](#)

[Unificazione](#)

[Conjunctive queries](#)

[Le Regole](#)

[Ragionamento](#)

[Derivazioni](#)

Overloading

Wildcards

Analisi di circuiti

Cammini su grafi

Cammini su grafi

Cammini su grafi

Cammini su grafi

Relational algebra

Negazione

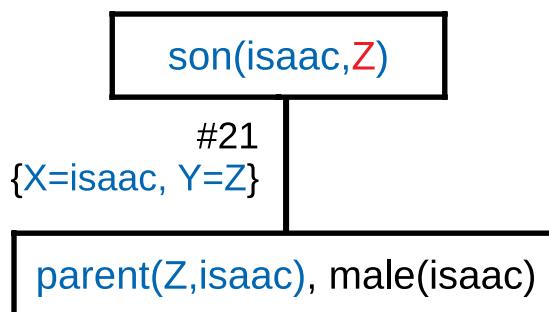
[Liste](#)

[Applicazioni](#)

Programmazione
nondeterministica

[Unicità di Prolog](#)

```
/* 4 */ father(abraham,isaac).      ....  
/* 8 */ mother(sarah,isaac).       ....  
/*13*/ male(isaac).             ....  
/*19*/ parent(X,Y) :- father(X,Y).  
/*20*/ parent(X,Y) :- mother(X,Y).  
/*21*/ son(X,Y)      :- parent(Y,X), male(X).
```





Esempio di derivazione

Paradigma logico

Prolog

Costrutti Prolog

Unificazione

Conjunctive queries

Le Regole

Ragionamento

Derivazioni

Overloading

Wildcards

Analisi di circuiti

Cammini su grafi

Cammini su grafi

Cammini su grafi

Cammini su grafi

Relational algebra

Negazione

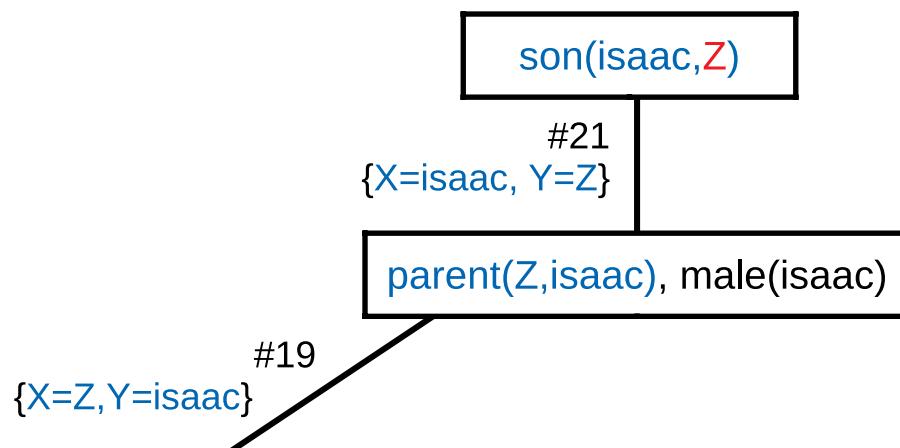
Liste

Applicazioni

Programmazione nondeterministica

Unicità di Prolog

```
/* 4 */ father(abraham,isaac).      ....  
/* 8 */ mother(sarah,isaac).       ....  
/*13*/ male(isaac).             ....  
/*19*/ parent(X,Y) :- father(X,Y).  
/*20*/ parent(X,Y) :- mother(X,Y).  
/*21*/ son(X,Y)      :- parent(Y,X), male(X).
```





Esempio di derivazione

Paradigma logico

Prolog

Costrutti Prolog

Unificazione

Conjunctive queries

Le Regole

Ragionamento

Derivazioni

Overloading

Wildcards

Analisi di circuiti

Cammini su grafi

Cammini su grafi

Cammini su grafi

Cammini su grafi

Relational algebra

Negazione

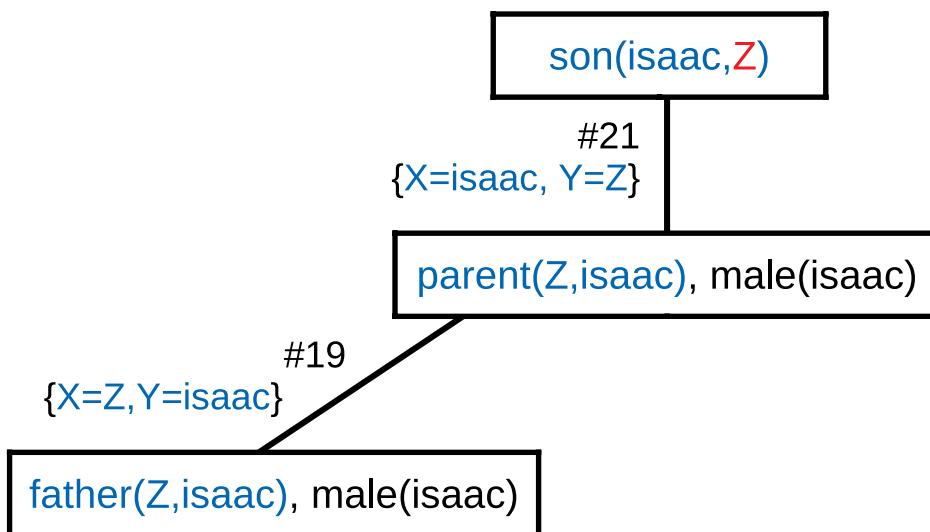
Liste

Applicazioni

Programmazione nondeterministica

Unicità di Prolog

```
/* 4 */ father(abraham,isaac).      ....  
/* 8 */ mother(sarah,isaac).       ....  
/*13*/ male(isaac).             ....  
/*19*/ parent(X,Y) :- father(X,Y).  
/*20*/ parent(X,Y) :- mother(X,Y).  
/*21*/ son(X,Y)      :- parent(Y,X), male(X).
```





Esempio di derivazione

Paradigma logico

Prolog

Costrutti Prolog

Unificazione

Conjunctive queries

Le Regole

Ragionamento

Derivazioni

Overloading

Wildcards

Analisi di circuiti

Cammini su grafi

Cammini su grafi

Cammini su grafi

Cammini su grafi

Relational algebra

Negazione

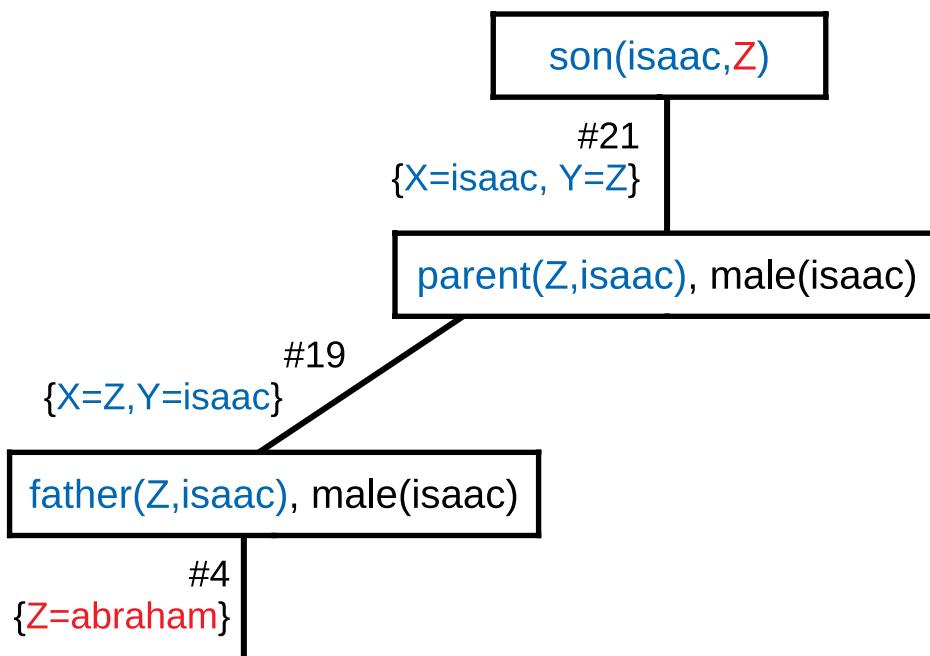
Liste

Applicazioni

Programmazione nondeterministica

Unicità di Prolog

```
/* 4 */ father(abraham,isaac).      ....  
/* 8 */ mother(sarah,isaac).       ....  
/*13*/ male(isaac).             ....  
/*19*/ parent(X,Y) :- father(X,Y).  
/*20*/ parent(X,Y) :- mother(X,Y).  
/*21*/ son(X,Y)      :- parent(Y,X), male(X).
```





Esempio di derivazione

Paradigma logico

Prolog

Costrutti Prolog

Unificazione

Conjunctive queries

Le Regole

Ragionamento

Derivazioni

Overloading

Wildcards

Analisi di circuiti

Cammini su grafi

Cammini su grafi

Cammini su grafi

Cammini su grafi

Relational algebra

Negazione

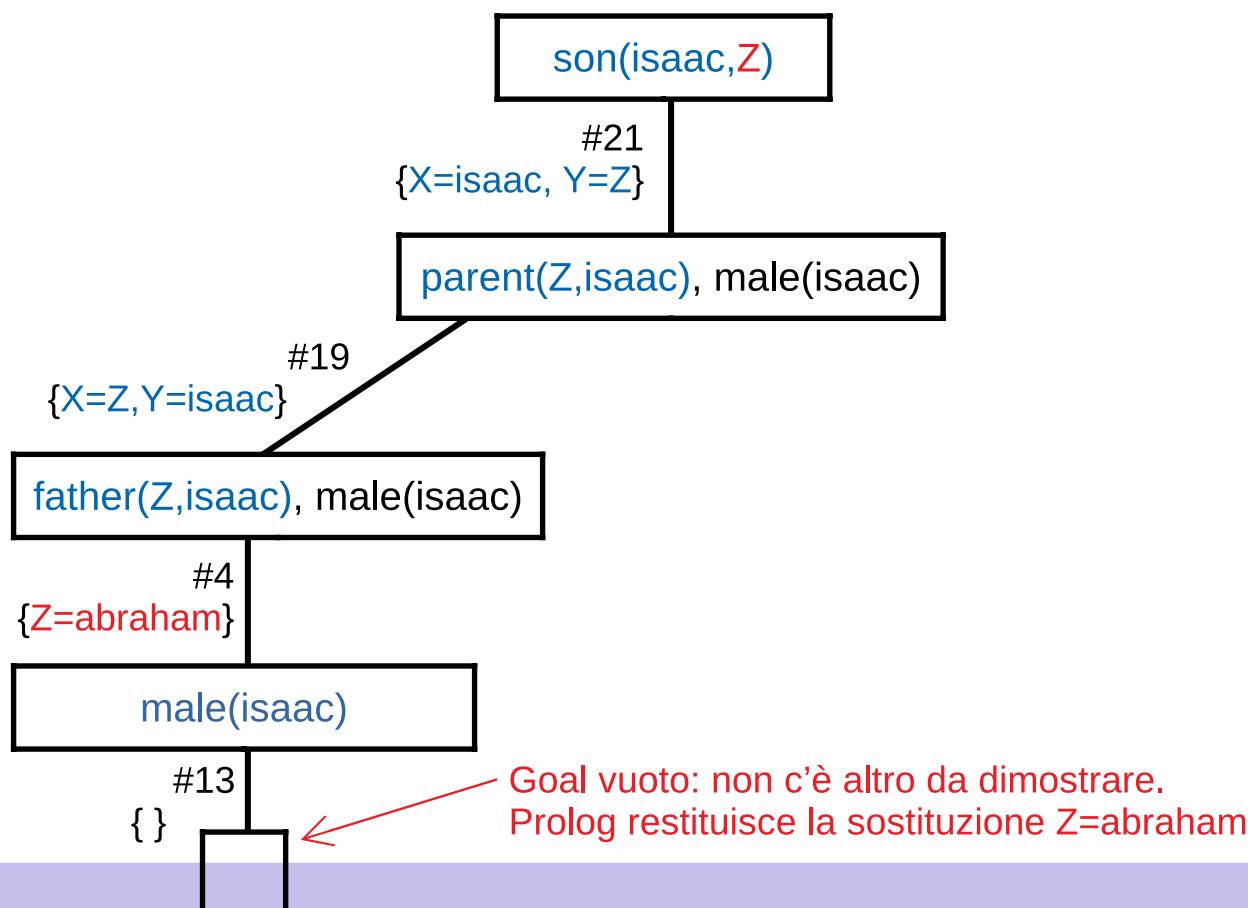
Liste

Applicazioni

Programmazione nondeterministica

Unicità di Prolog

```
/* 4 */ father(abraham,isaac).      ....  
/* 8 */ mother(sarah,isaac).       ....  
/*13*/ male(isaac).             ....  
/*19 */ parent(X,Y) :- father(X,Y).  
/*20 */ parent(X,Y) :- mother(X,Y).  
/*21 */ son(X,Y)      :- parent(Y,X), male(X).
```





Esempio di derivazione

Paradigma logico

Prolog

Costrutti Prolog

Unificazione

Conjunctive queries

Le Regole

Ragionamento

Derivazioni

Overloading

Wildcards

Analisi di circuiti

Cammini su grafi

Cammini su grafi

Cammini su grafi

Cammini su grafi

Relational algebra

Negazione

Liste

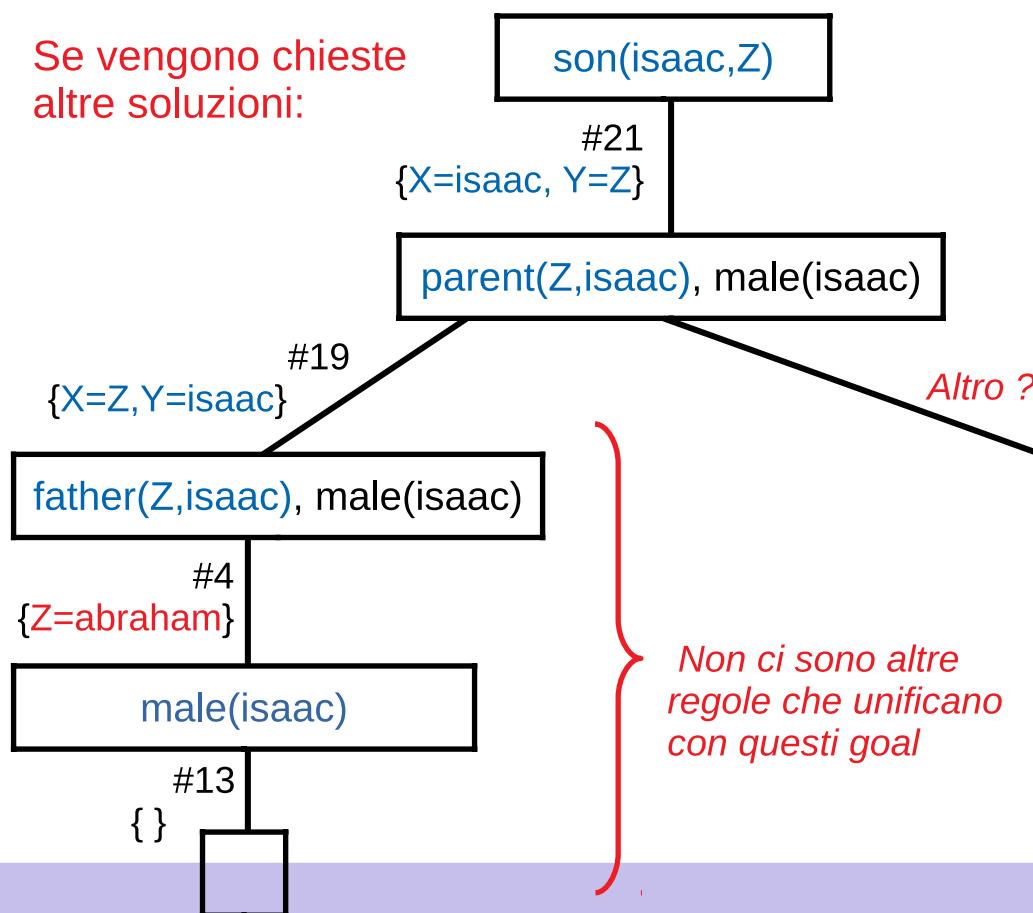
Applicazioni

Programmazione nondeterministica

Unicità di Prolog

```
/* 4 */ father(abraham,isaac).      ....  
/* 8 */ mother(sarah,isaac).       ....  
/*13*/ male(isaac).             ....  
/*19 */ parent(X,Y) :- father(X,Y).  
/*20 */ parent(X,Y) :- mother(X,Y).  
/*21 */ son(X,Y)      :- parent(Y,X), male(X).
```

Se vengono chieste
altre soluzioni:





Esempio di derivazione

Paradigma logico

Prolog

Costrutti Prolog

Unificazione

Conjunctive queries

Le Regole

Ragionamento

Derivazioni

Overloading

Wildcards

Analisi di circuiti

Cammini su grafi

Cammini su grafi

Cammini su grafi

Cammini su grafi

Relational algebra

Negazione

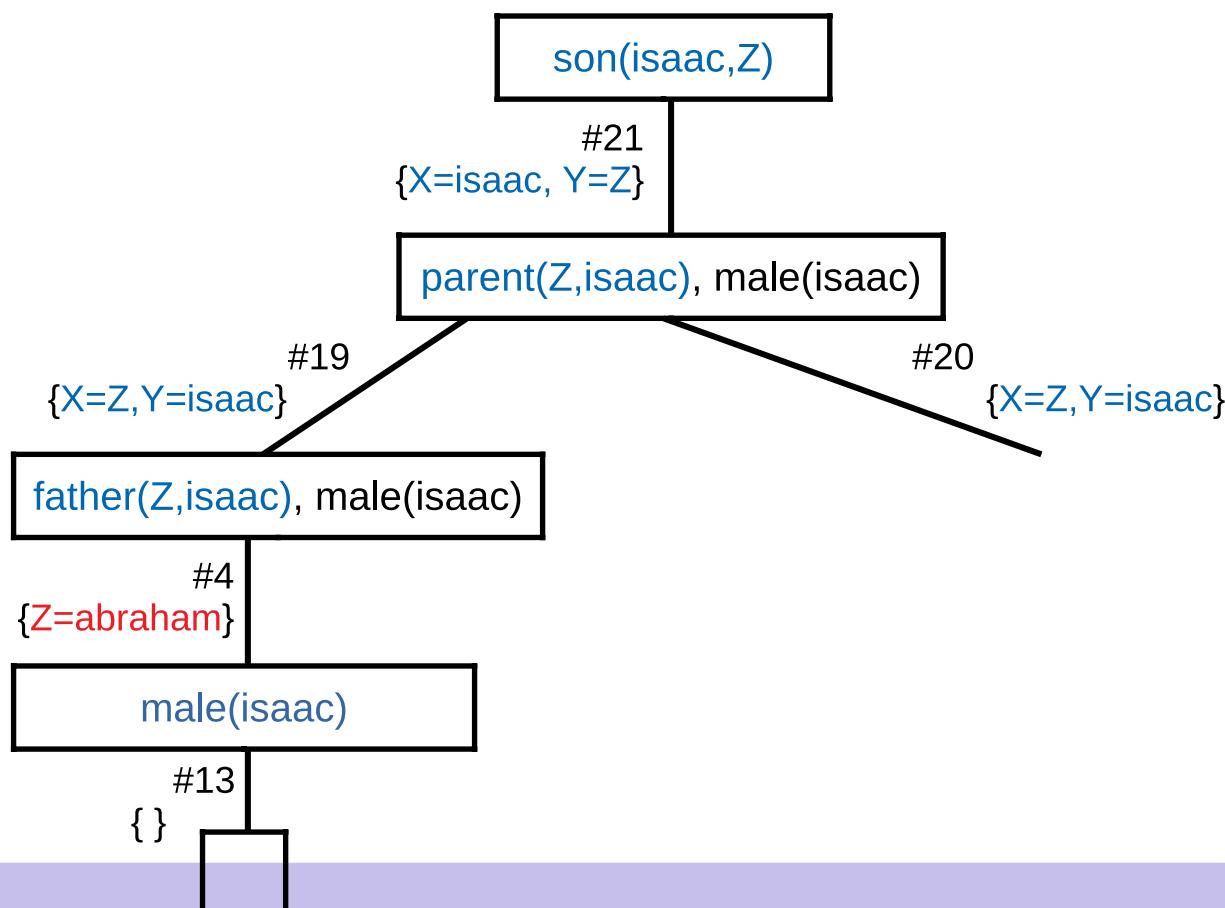
Liste

Applicazioni

Programmazione nondeterministica

Unicità di Prolog

```
/* 4 */ father(abraham,isaac).      ....  
/* 8 */ mother(sarah,isaac).       ....  
/*13*/ male(isaac).             ....  
/*19*/ parent(X,Y) :- father(X,Y).  
/*20*/ parent(X,Y) :- mother(X,Y).  
/*21*/ son(X,Y)      :- parent(Y,X), male(X).
```





Esempio di derivazione

Paradigma logico

Prolog

Costrutti Prolog

Unificazione

Conjunctive queries

Le Regole

Ragionamento

Derivazioni

Overloading

Wildcards

Analisi di circuiti

Cammini su grafi

Cammini su grafi

Cammini su grafi

Cammini su grafi

Relational algebra

Negazione

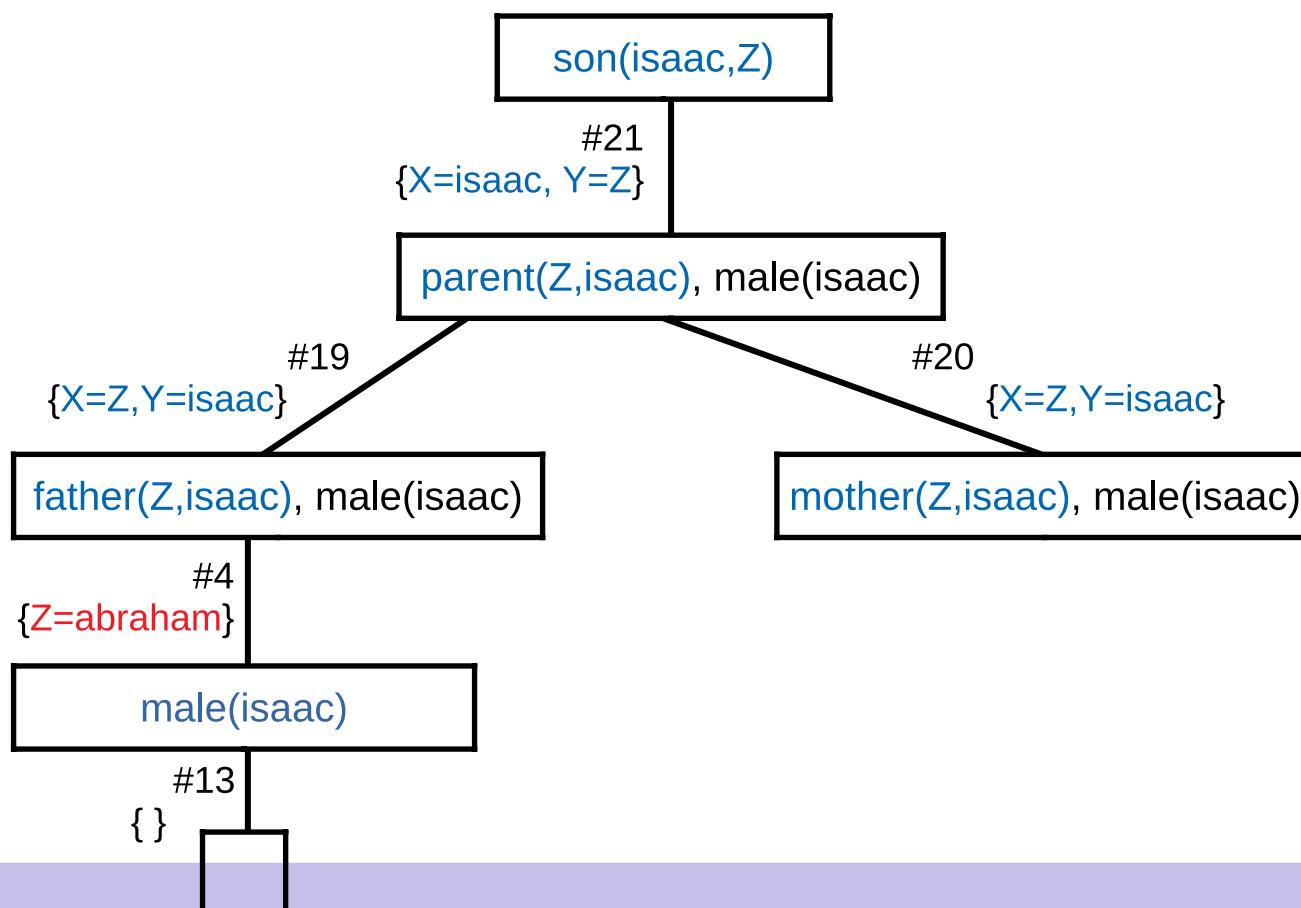
Liste

Applicazioni

Programmazione nondeterministica

Unicità di Prolog

```
/* 4 */ father(abraham,isaac).      ....  
/* 8 */ mother(sarah,isaac).       ....  
/*13*/ male(isaac).             ....  
/*19*/ parent(X,Y) :- father(X,Y).  
/*20*/ parent(X,Y) :- mother(X,Y).  
/*21*/ son(X,Y)      :- parent(Y,X), male(X).
```





Esempio di derivazione

Paradigma logico

Prolog

Costrutti Prolog

Unificazione

Conjunctive queries

Le Regole

Ragionamento

Derivazioni

Overloading

Wildcards

Analisi di circuiti

Cammini su grafi

Cammini su grafi

Cammini su grafi

Cammini su grafi

Relational algebra

Negazione

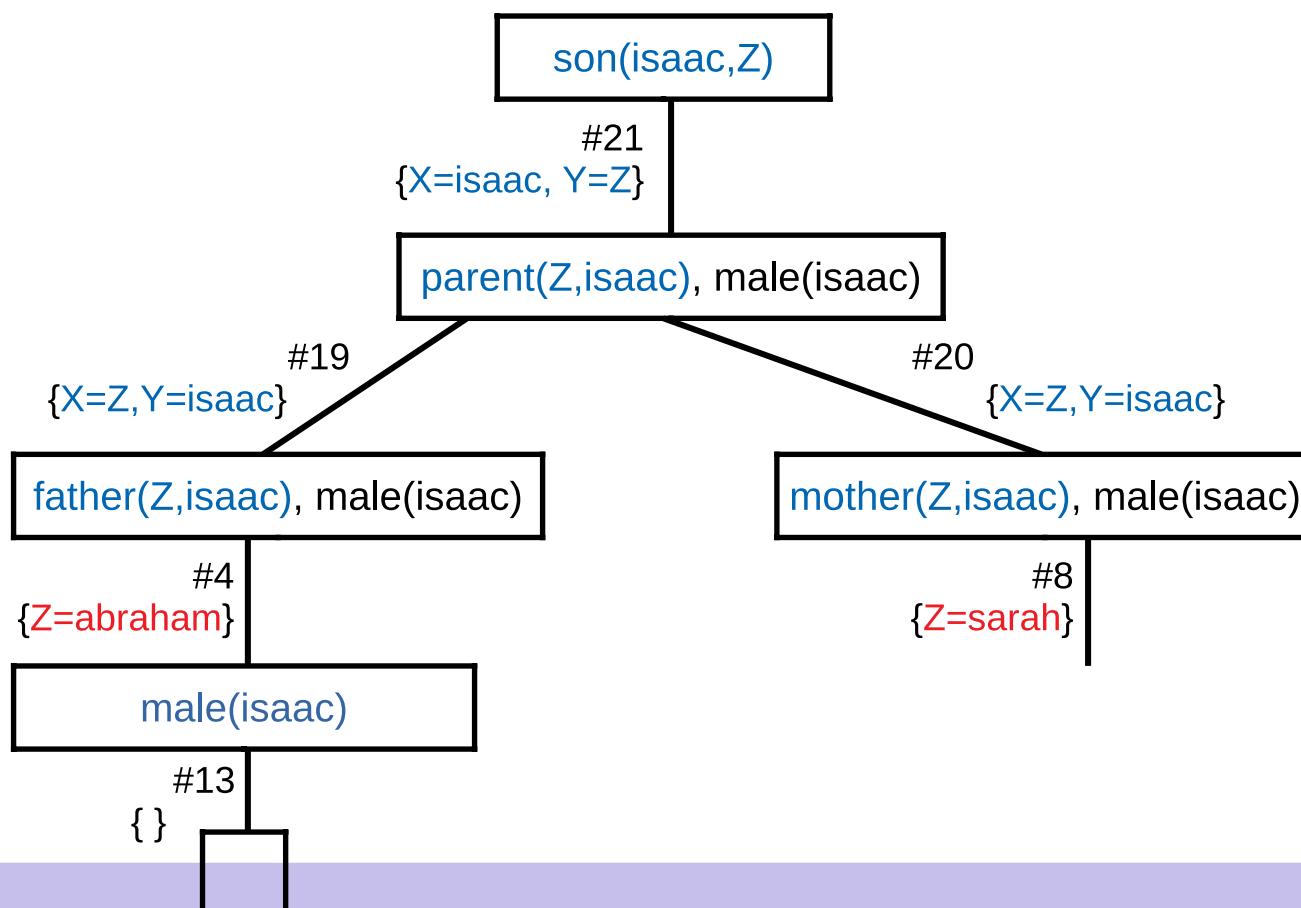
Liste

Applicazioni

Programmazione nondeterministica

Unicità di Prolog

```
/* 4 */ father(abraham,isaac).      ....  
/* 8 */ mother(sarah,isaac).       ....  
/*13*/ male(isaac).             ....  
/*19*/ parent(X,Y) :- father(X,Y).  
/*20*/ parent(X,Y) :- mother(X,Y).  
/*21*/ son(X,Y)      :- parent(Y,X), male(X).
```





Esempio di derivazione

Paradigma logico

Prolog

Costrutti Prolog

Unificazione

Conjunctive queries

Le Regole

Ragionamento

Derivazioni

Overloading

Wildcards

Analisi di circuiti

Cammini su grafi

Cammini su grafi

Cammini su grafi

Cammini su grafi

Relational algebra

Negazione

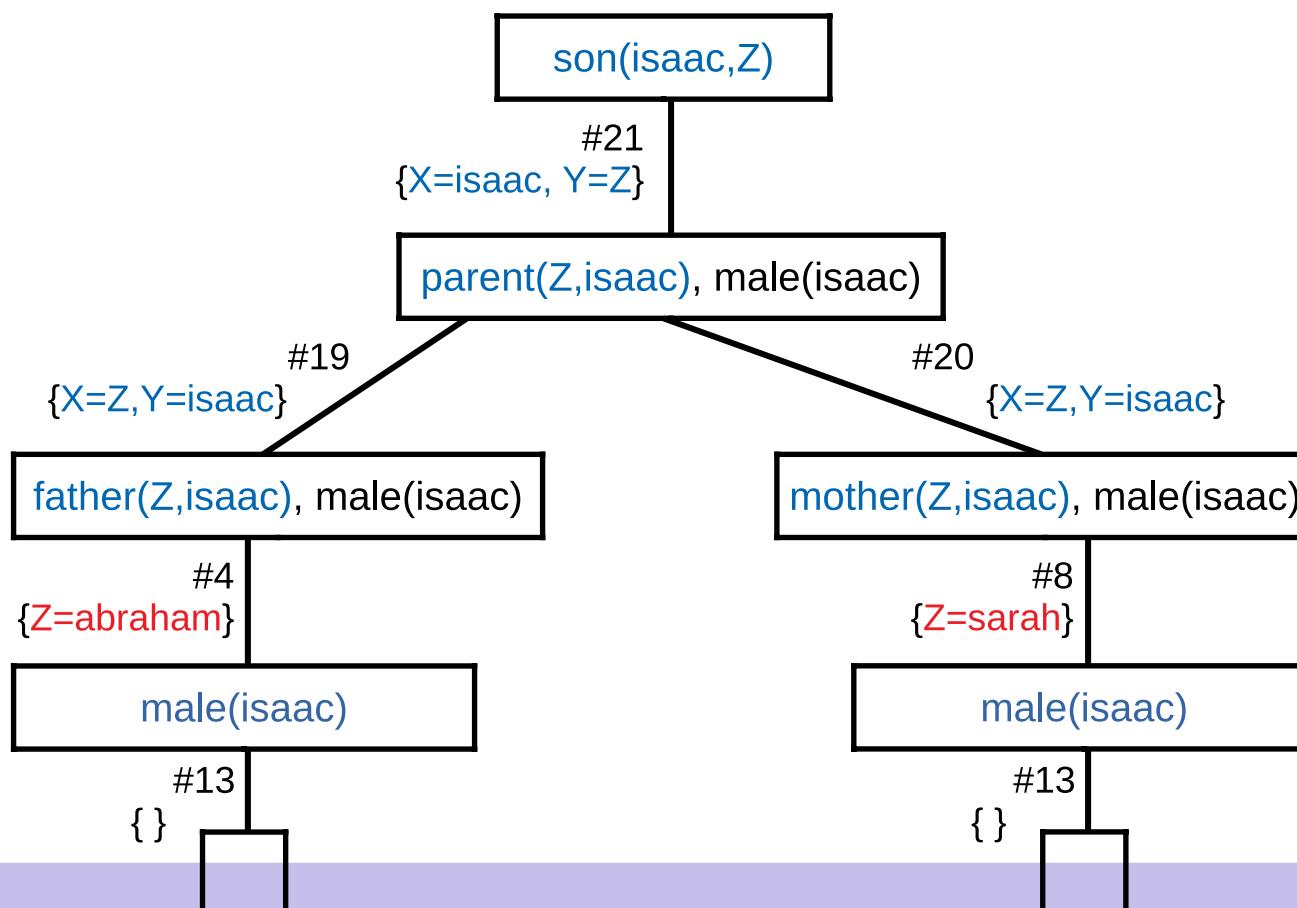
Liste

Applicazioni

Programmazione nondeterministica

Unicità di Prolog

```
/* 4 */ father(abraham,isaac).      ....  
/* 8 */ mother(sarah,isaac).       ....  
/*13*/ male(isaac).             ....  
/*19 */ parent(X,Y) :- father(X,Y).  
/*20 */ parent(X,Y) :- mother(X,Y).  
/*21 */ son(X,Y)      :- parent(Y,X), male(X).
```





Esempio di derivazione con standardization apart

Paradigma logico

Prolog

Costrutti Prolog

Unificazione

Conjunctive queries

Le Regole

Ragionamento

Derivazioni

Overloading

Wildcards

Analisi di circuiti

Cammini su grafi

Cammini su grafi

Cammini su grafi

Cammini su grafi

Relational algebra

Negazione

Liste

Applicazioni

Programmazione nondeterministica

Unicità di Prolog

```
/* 4 */ father(abraham,isaac).      ....  
/* 8 */ mother(sarah,isaac).       ....  
/*13*/ male(isaac).             ....  
/*19 */ parent(X,Y) :- father(X,Y).  
/*20 */ parent(X,Y) :- mother(X,Y).  
/*21 */ son(X,Y)      :- parent(Y,X), male(X).
```

Cosa succede
davvero

son(isaac,Z)

#21
{X₁=isaac, Y₁=Z}

parent(Z,isaac), male(isaac)

Le variabili delle
regole vengono
ridenominate

#19
{X₂=Z, Y₂=isaac}

father(Z,isaac), male(isaac)

#20
{X₃=Z, Y₃=isaac}

mother(Z,isaac), male(isaac)

#4
{Z=abraham}

male(isaac)

#8
{Z=sarah}

male(isaac)

#13
{}



#13
{}





Overloading

[Paradigma logico](#)

[Prolog](#)

[Costrutti Prolog](#)

[Unificazione](#)

[Conjunctive queries](#)

[Le Regole](#)

[Ragionamento](#)

[Derivazioni](#)

[Overloading](#)

[Wildcards](#)

[Analisi di circuiti](#)

[Cammini su grafi](#)

[Cammini su grafi](#)

[Cammini su grafi](#)

[Cammini su grafi](#)

[Relational algebra](#)

[Negazione](#)

[Liste](#)

[Applicazioni](#)

[Programmazione nondeterministica](#)

[Unicità di Prolog](#)

- Si può usare lo stesso nome di predicato con numeri diversi di argomenti
- Simili prediciati vengono trattati come prediciati diversi
- Esempio: dall'informazione che "X è madre di Y" derivare il concetto di essere madre

```
mother(Mom) :- mother(Mom,X).  
/* Mom è una mamma se esiste un X tale che Mom è madre di X */  
/* Notare che abbiamo un mother unario e uno binario */
```



Wildcards

- Notare che nel goal `mother(Mom, X)` il valore di `X` non interessa
- In questi casi possiamo usare wildcards simili a ML

```
mother(Mom) :- mother(Mom, _).
```

Paradigma logico

Prolog

Costrutti Prolog

Unificazione

Conjunctive queries

Le Regole

Ragionamento

Derivazioni

Overloading

Wildcards

Analisi di circuiti

Cammini su grafi

Cammini su grafi

Cammini su grafi

Cammini su grafi

Relational algebra

Negazione

Liste

Applicazioni

Programmazione nondeterministica

Unicità di Prolog



Wildcards

Paradigma logico

Prolog

Costrutti Prolog

Unificazione

Conjunctive queries

Le Regole

Ragionamento

Derivazioni

Overloading

Wildcards

Analisi di circuiti

Cammini su grafi

Cammini su grafi

Cammini su grafi

Cammini su grafi

Relational algebra

Negazione

Liste

Applicazioni

Programmazione nondeterministica

Unicità di Prolog

- Notare che nel goal `mother(Mom, X)` il valore di `X` non interessa
- In questi casi possiamo usare wildcards simili a ML

```
mother(Mom) :- mother(Mom, _).
```

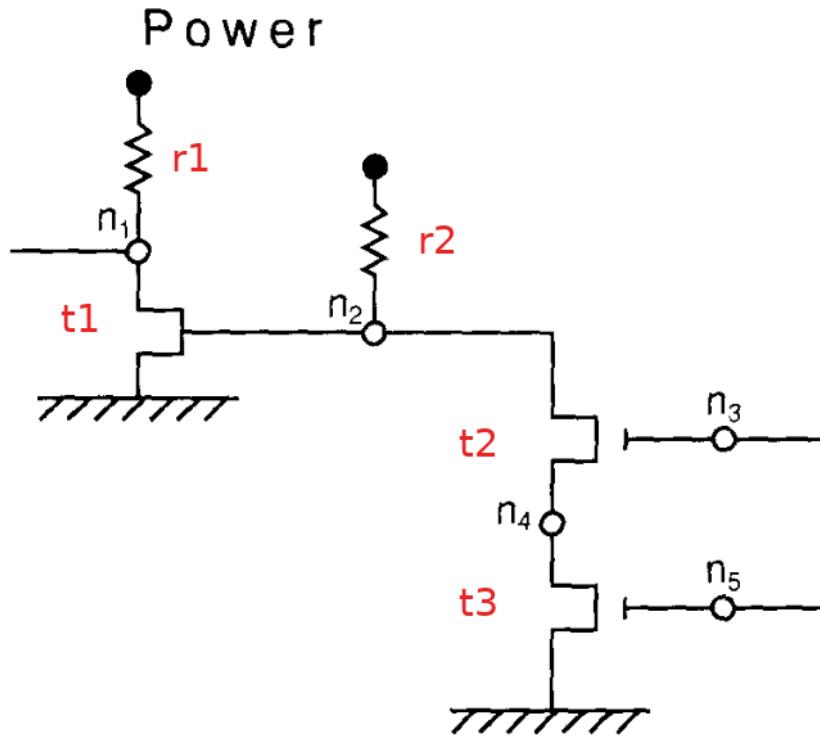
- Attenzione: se usiamo due volte una variabile (ad es. `X`) allora in quei punti devo avere lo stesso valore, mentre ogni wildcard può essere associata a un valore diverso

```
?- mother(X, X).  
false      (nessuna è madre di sè stessa)  
  
?- mother(_, _).  
true       (cerca un fatto mother(X, Y) nel database)
```



Esempio: Analisi di Circuiti

Con i fatti possiamo descrivere circuiti



```
resistor(r1,power,n1).  
resistor(r2,power,n2).  
transistor(t1,n2,ground,n1).  
transistor(t2,n3,n4,n2).  
transistor(t3,n5,ground,n4).
```

Figure 2.2 A logical circuit

Paradigma logico

Prolog

Costrutti Prolog

Unificazione

Conjunctive queries

Le Regole

Ragionamento

Derivazioni

Overloading

Wildcards

Analisi di circuiti

Cammini su grafi

Cammini su grafi

Cammini su grafi

Cammini su grafi

Relational algebra

Negazione

Liste

Applicazioni

Programmazione
nondeterministica

Unicità di Prolog



Esempio: Analisi di Circuiti

Paradigma logico

Prolog

Costrutti Prolog

Unificazione

Conjunctive queries

Le Regole

Ragionamento

Derivazioni

Overloading

Wildcards

Analisi di circuiti

Cammini su grafi

Cammini su grafi

Cammini su grafi

Cammini su grafi

Relational algebra

Negazione

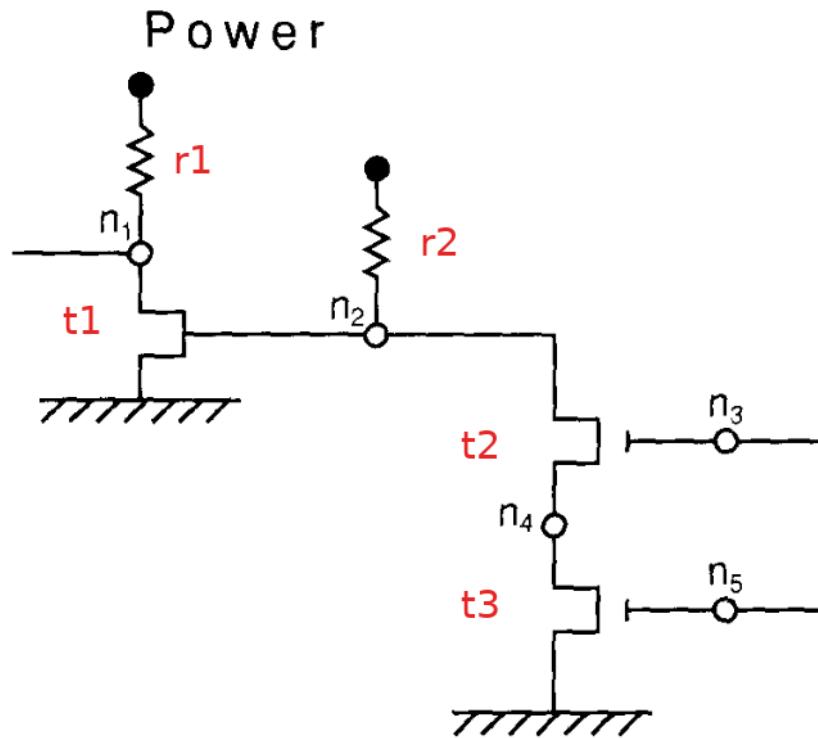
Liste

Applicazioni

Programmazione
nondeterministica

Unicità di Prolog

Con i fatti possiamo descrivere circuiti



```
resistor(r1, power, n1).  
resistor(r2, power, n2).  
transistor(t1, n2, ground, n1).  
transistor(t2, n3, n4, n2).  
transistor(t3, n5, ground, n4).
```

Figure 2.2 A logical circuit

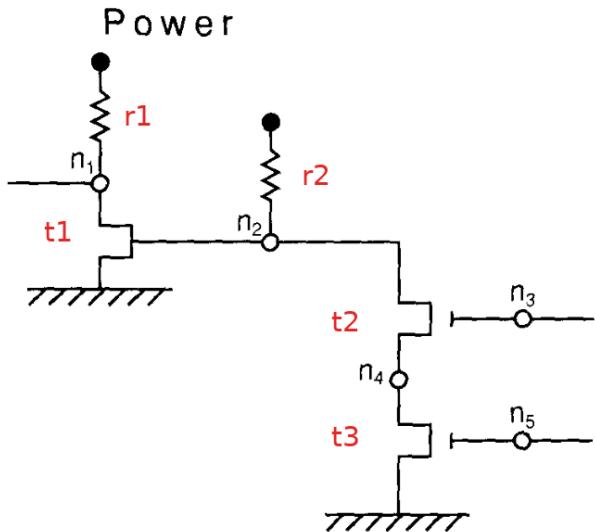
Da questa descrizione fisica vogliamo derivarne una funzionale

- ovvero: cosa fanno questi componenti?



Esempio: Analisi di Circuiti

■ Descrizione funzionale dei circuiti



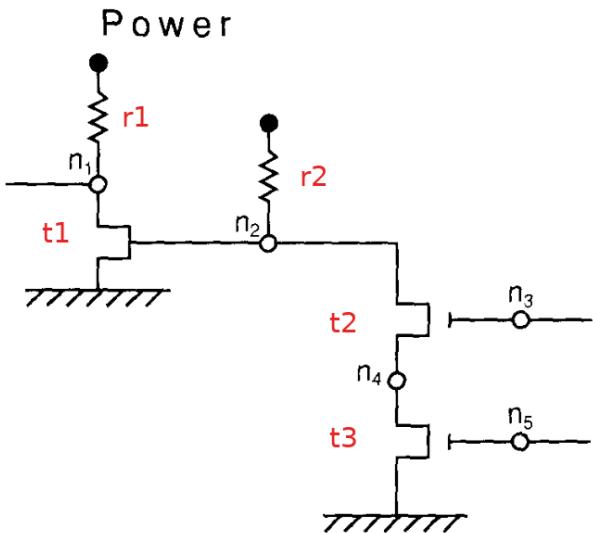
```
inverter( inv(T,R), In, Out ) :-  
    transistor(T, In, ground, Out ),  
    resistor(R, power, Out ).
```

Figure 2.2 A logical circuit



Esempio: Analisi di Circuiti

■ Descrizione funzionale dei circuiti



```
inverter( inv(T,R), In, Out ) :-  
    transistor(T, In, ground, Out ),  
    resistor(R, power, Out ).  
  
nand_gate( nand(T1,T2,R), In1, In2, Out ) :-  
    transistor(T1, In1, X, Out),  
    transistor(T2, In2, ground, X),  
    resistor(R,power,Out).
```

Figure 2.2 A logical circuit



Esempio: Analisi di Circuiti

■ Descrizione funzionale dei circuiti

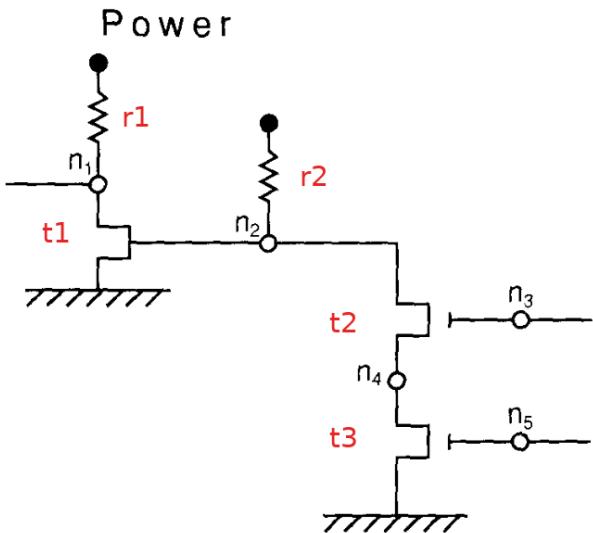


Figure 2.2 A logical circuit

```
inverter( inv(T,R), In, Out ) :-  
    transistor(T, In, ground, Out ),  
    resistor(R, power, Out ).  
  
nand_gate( nand(T1,T2,R), In1, In2, Out ) :-  
    transistor(T1, In1, X, Out),  
    transistor(T2, In2, ground, X),  
    resistor(R,power,Out).  
  
and_gate( and(N,I), In1, In2, Out ) :-  
    nand_gate(N, In1, In2, X),  
    inverter(I,X,Out).
```



Esempio: Analisi di Circuiti

■ Descrizione funzionale dei circuiti

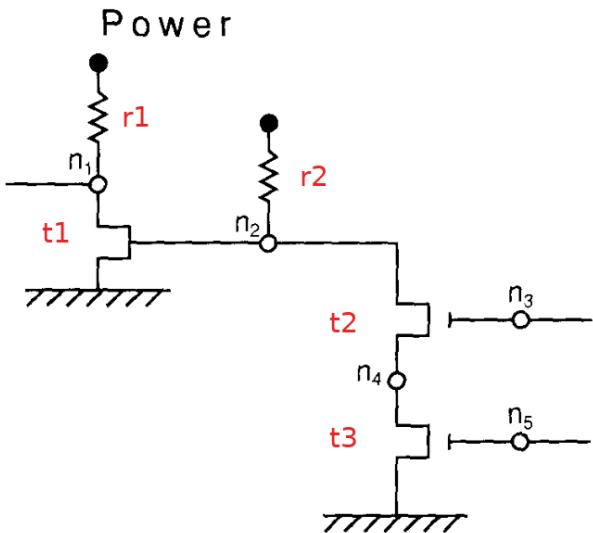


Figure 2.2 A logical circuit

```
inverter( inv(T,R), In, Out ) :-  
    transistor(T, In, ground, Out ),  
    resistor(R, power, Out ).  
  
nand_gate( nand(T1,T2,R), In1, In2, Out ) :-  
    transistor(T1, In1, X, Out),  
    transistor(T2, In2, ground, X),  
    resistor(R,power,Out).  
  
and_gate( and(N,I), In1, In2, Out ) :-  
    nand_gate(N, In1, In2, X),  
    inverter(I,X,Out).
```

Ora possiamo trovare i componenti logici implementati dal circuito

```
?- inverter(X,Y,Z).  
X = inv(t1, r1),  
Y = n2,  
Z = n1.
```



Esempio: Analisi di Circuiti

■ Descrizione funzionale dei circuiti

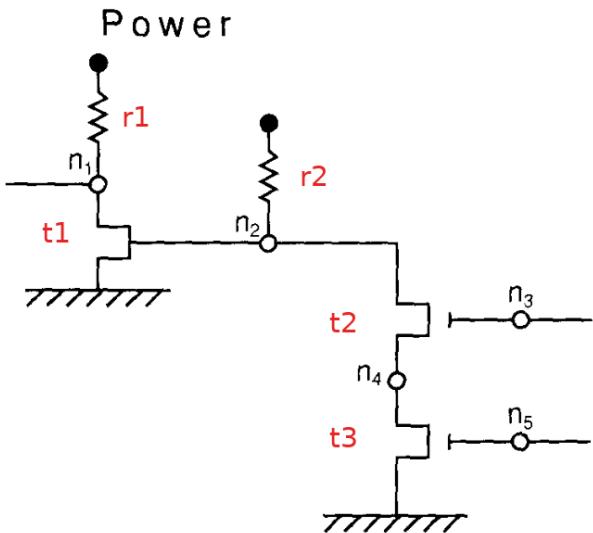


Figure 2.2 A logical circuit

```
inverter( inv(T,R), In, Out ) :-  
    transistor(T, In, ground, Out ),  
    resistor(R, power, Out ).  
  
nand_gate( nand(T1,T2,R), In1, In2, Out ) :-  
    transistor(T1, In1, X, Out),  
    transistor(T2, In2, ground, X),  
    resistor(R,power,Out).  
  
and_gate( and(N,I), In1, In2, Out ) :-  
    nand_gate(N, In1, In2, X),  
    inverter(I,X,Out).
```

Ora possiamo trovare i componenti logici implementati dal circuito

```
?- inverter(X,Y,Z).  
X = inv(t1, r1),  
Y = n2,  
Z = n1.
```

```
?- nand_gate(X,Y,Z,0).  
X = nand(t2, t3, r2),  
Y = n3,  
Z = n5,  
0 = n2.
```



Esempio: Analisi di Circuiti

■ Descrizione funzionale dei circuiti

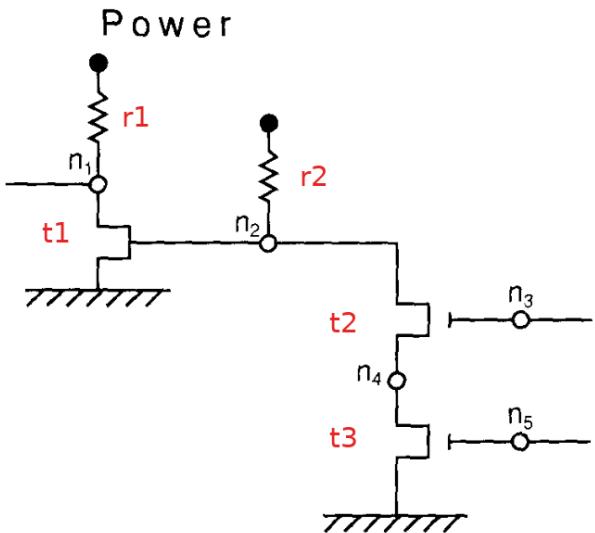


Figure 2.2 A logical circuit

```
inverter( inv(T,R), In, Out ) :-  
    transistor(T, In, ground, Out ),  
    resistor(R, power, Out ).  
  
nand_gate( nand(T1,T2,R), In1, In2, Out ) :-  
    transistor(T1, In1, X, Out),  
    transistor(T2, In2, ground, X),  
    resistor(R,power,Out).  
  
and_gate( and(N,I), In1, In2, Out ) :-  
    nand_gate(N, In1, In2, X),  
    inverter(I,X,Out).
```

Ora possiamo trovare i componenti logici implementati dal circuito

```
?- inverter(X,Y,Z).  
X = inv(t1, r1),  
Y = n2,  
Z = n1.
```

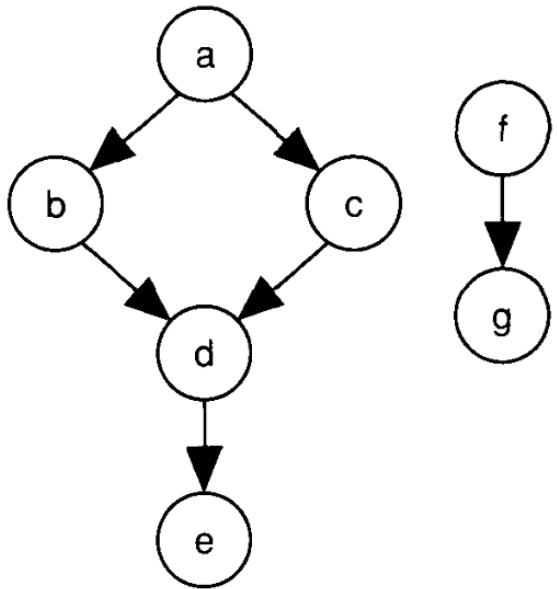
```
?- nand_gate(X,Y,Z,0).  
X = nand(t2, t3, r2),  
Y = n3,  
Z = n5,  
0 = n2.
```

```
?- and_gate(X,Y,Z,0).  
X =  
    and(nand(t2,t3,r2),inv(t1,r1)),  
Y = n3,  
Z = n5,  
0 = n1.
```



Regole ricorsive

Cerchiamo ora i cammini in un grafo



```
/* programma che descrive il grafo */

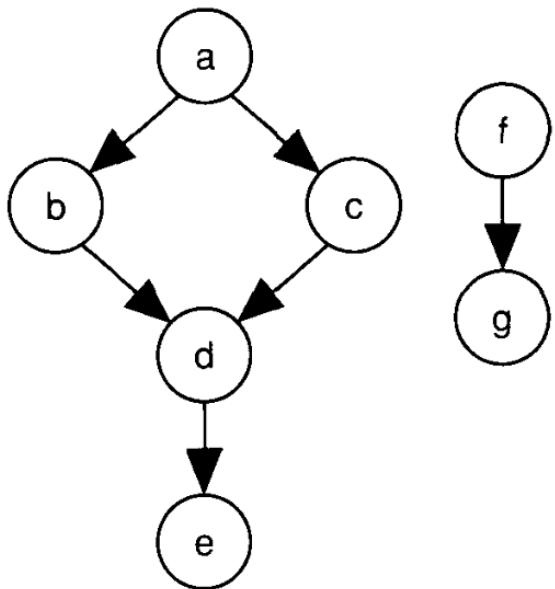
arco(a,b).           arco(b,d).           arco(d,e).
arco(a,c).           arco(c,d).           arco(f,g).
```

Figure 2.4 A simple graph



Regole ricorsive

Cerchiamo ora i cammini in un grafo



```
/* programma che descrive il grafo */

arco(a,b).           arco(b,d).           arco(d,e).
arco(a,c).           arco(c,d).           arco(f,g).
```

Verificare se due nodi sono connessi:

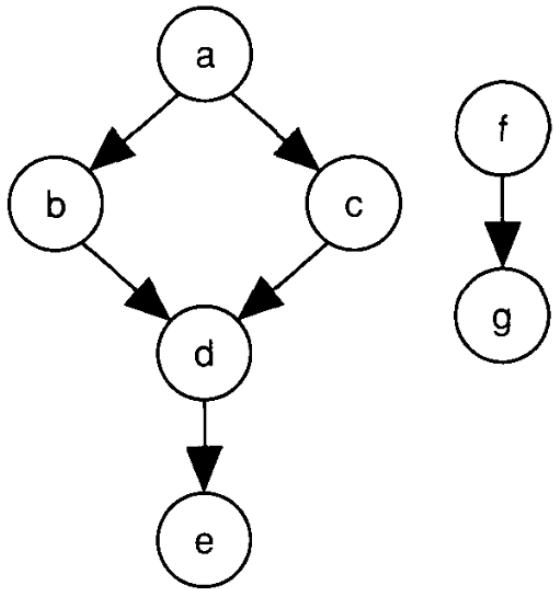
```
connessi(X,X).
```

Figure 2.4 A simple graph



Regole ricorsive

Cerchiamo ora i cammini in un grafo



```
/* programma che descrive il grafo */
arco(a,b).           arco(b,d).           arco(d,e).
arco(a,c).           arco(c,d).           arco(f,g).
```

Verificare se due nodi sono connessi:

```
connessi(X,X).
connessi(X,Y) :- arco(X,Z), connessi(Z,Y).
```

Figure 2.4 A simple graph



Search tree parziale per la query `connessi(a,d)`

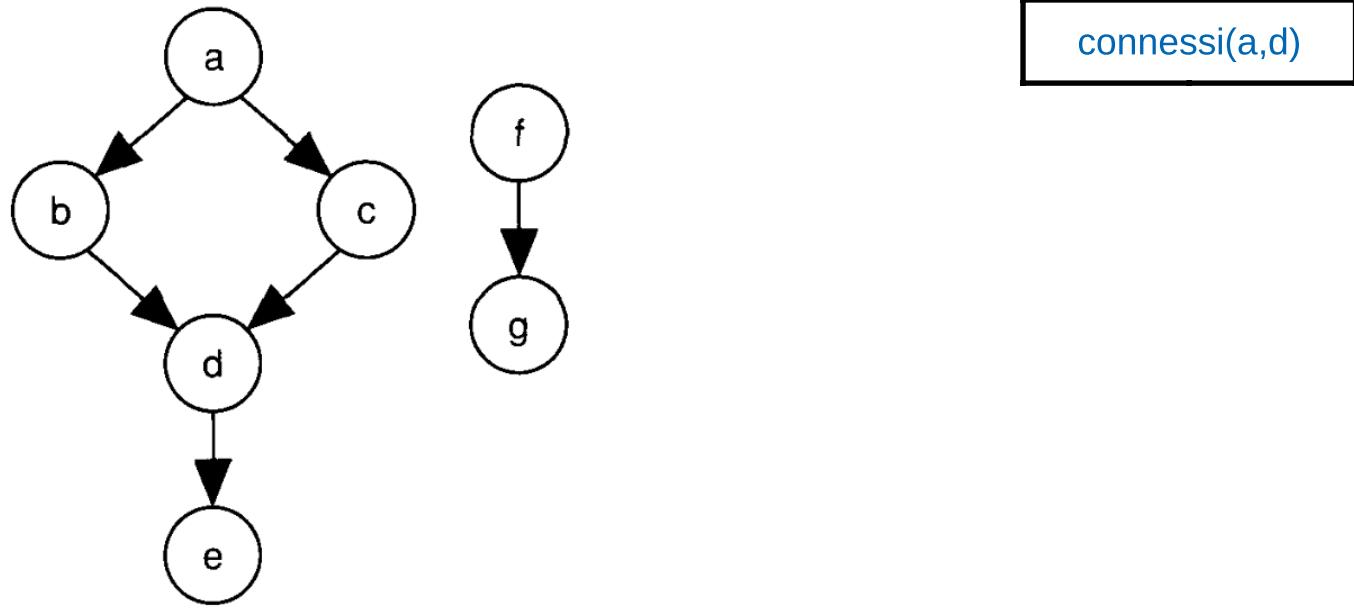


Figure 2.4 A simple graph



Search tree parziale per la query $\text{connessi}(a,d)$

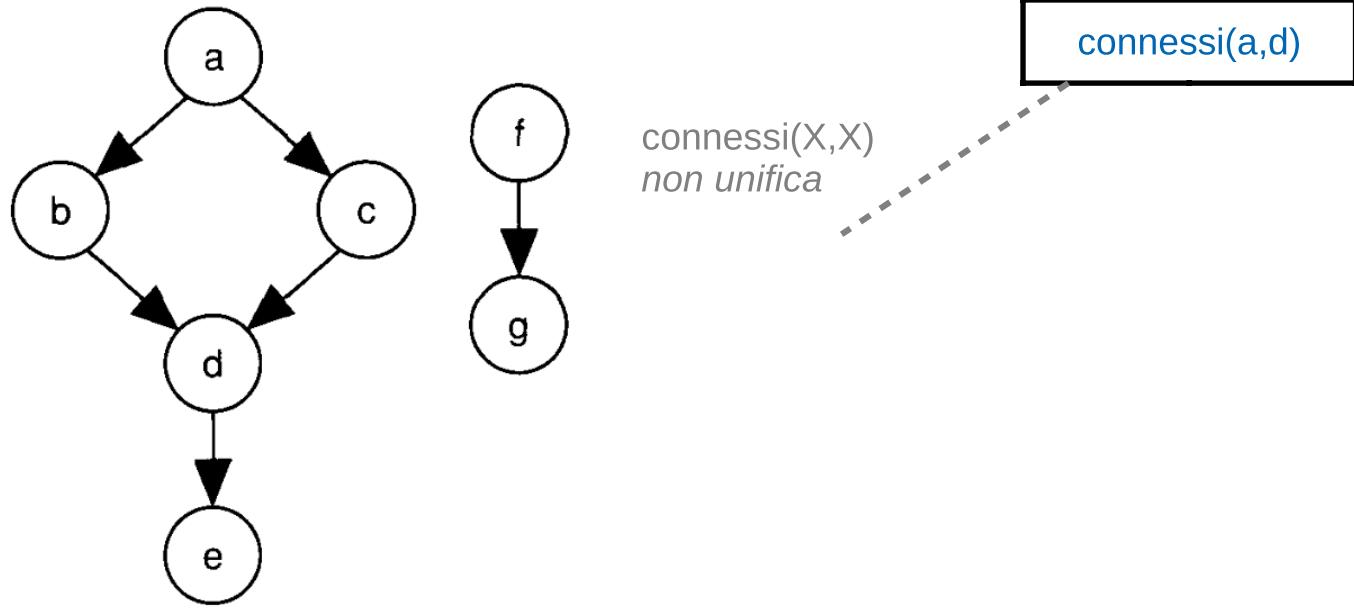
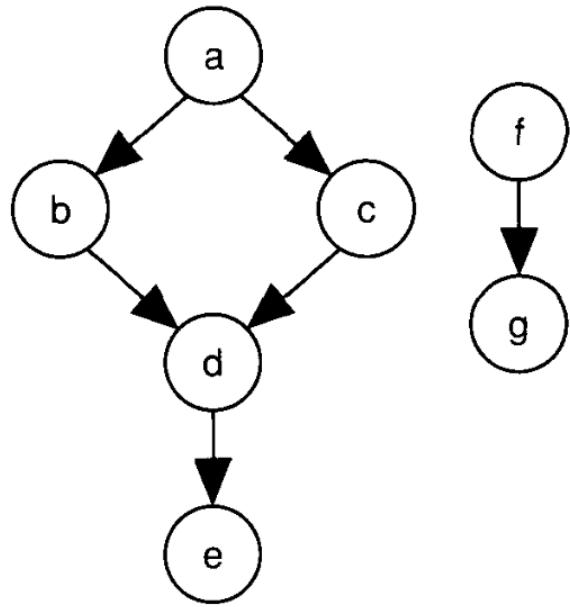


Figure 2.4 A simple graph



Search tree parziale per la query `connessi(a,d)`



`connessi(X,X)`
non unifica

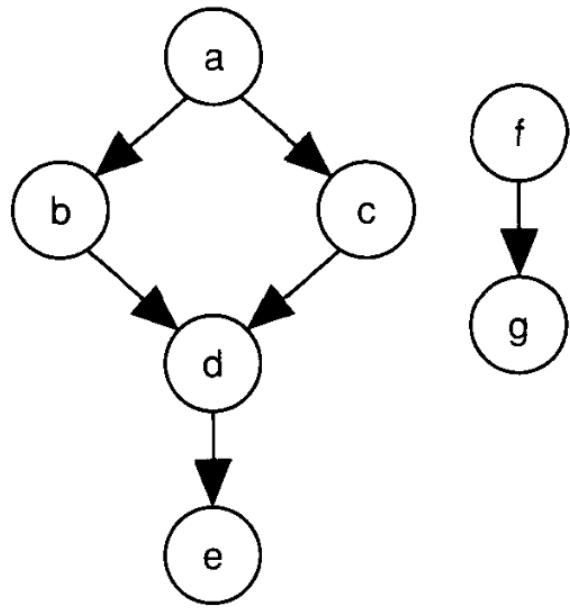
`connessi(a,d)`

`connessi(a,d) :- arco(a,Z1), connessi(Z1,d).`
 $\{ X_1=a , Y_1=d \}$

Figure 2.4 A simple graph



Search tree parziale per la query `connessi(a,d)`



`connessi(X,X)`
non unifica

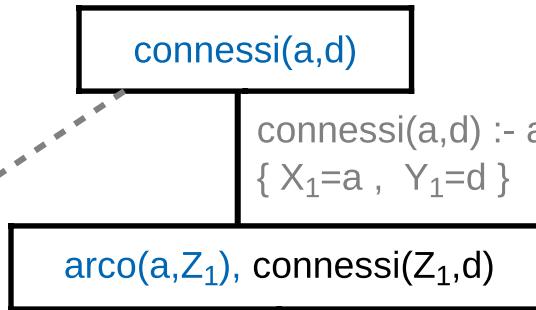
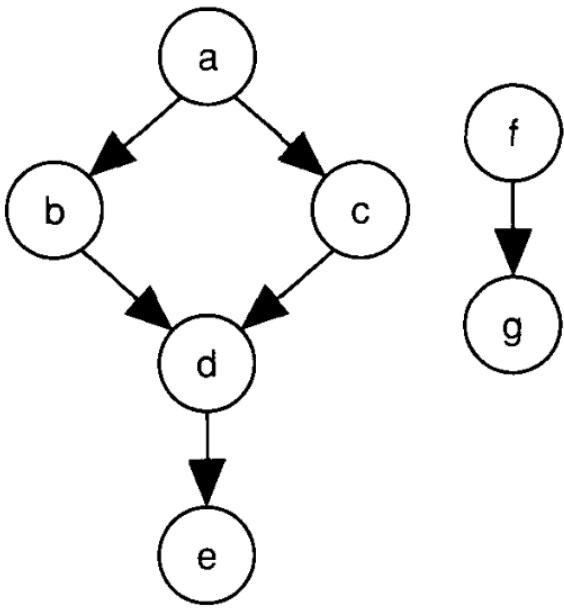


Figure 2.4 A simple graph



Search tree parziale per la query `connessi(a,d)`



`connessi(X,X)`
non unifica

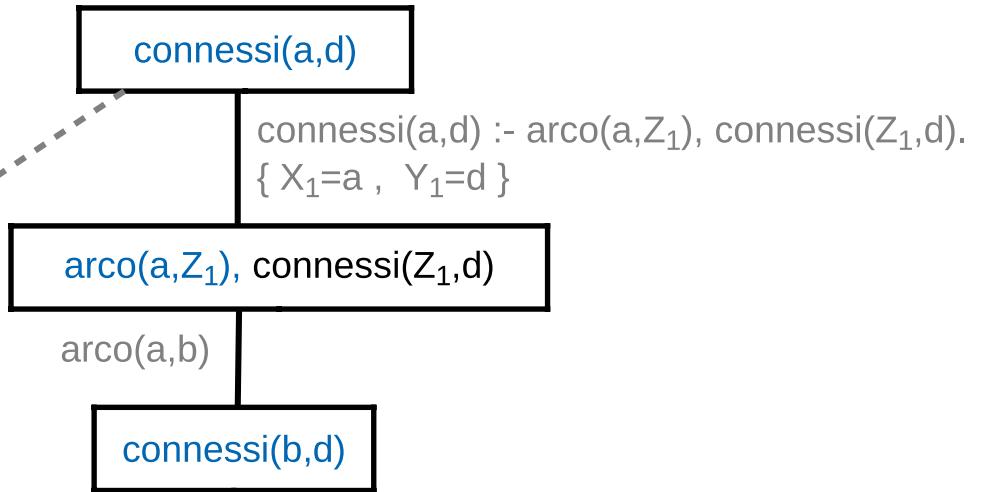
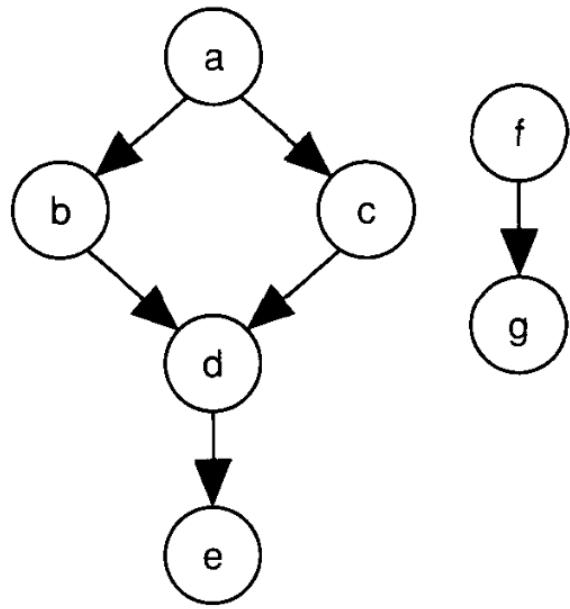


Figure 2.4 A simple graph



Search tree parziale per la query `connessi(a,d)`



`connessi(X,X)`
non unifica

`connessi(X,X)`
non unifica

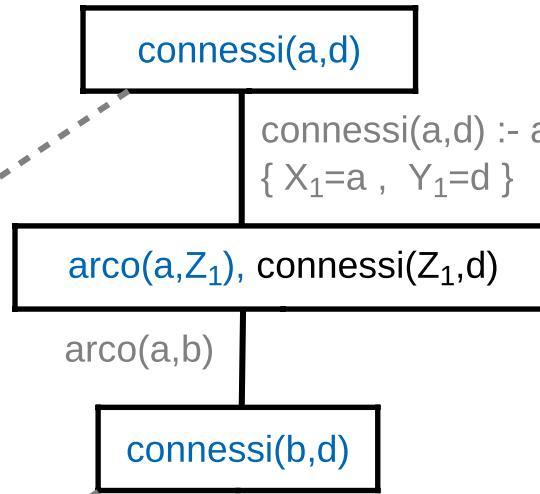
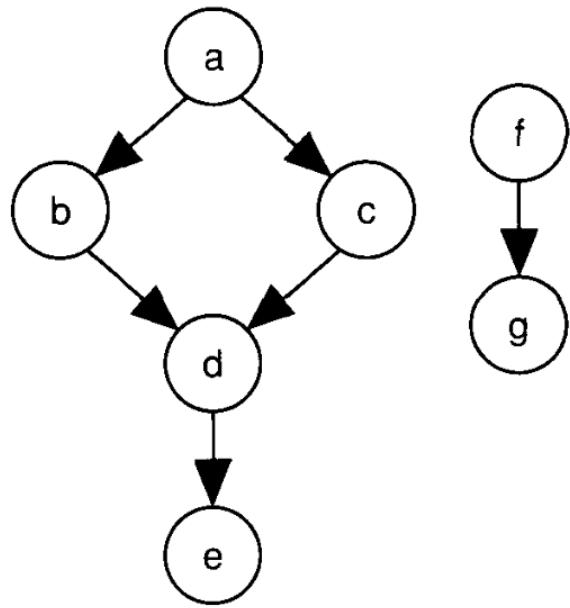


Figure 2.4 A simple graph



Search tree parziale per la query `connessi(a,d)`



*connessi(X,X)
non unifica*

*connessi(X,X)
non unifica*

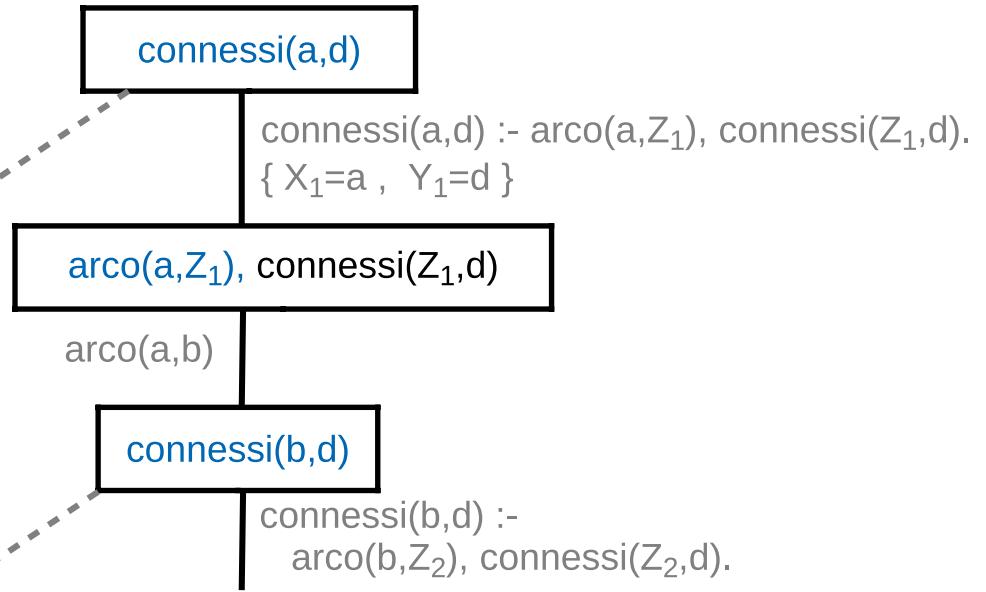


Figure 2.4 A simple graph



Search tree parziale per la query `connessi(a,d)`

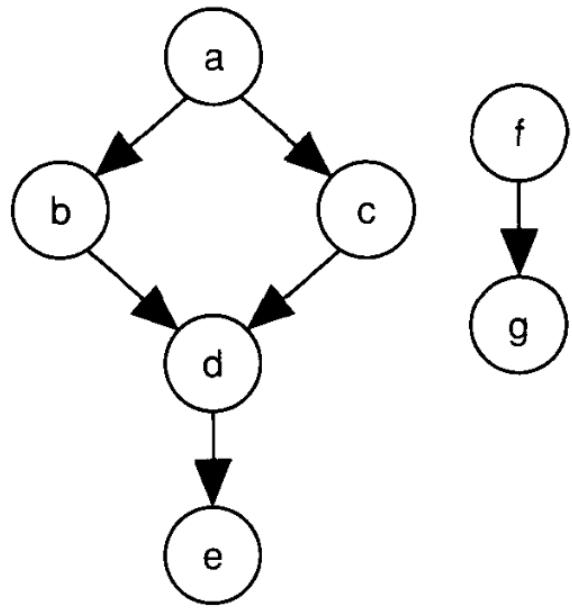
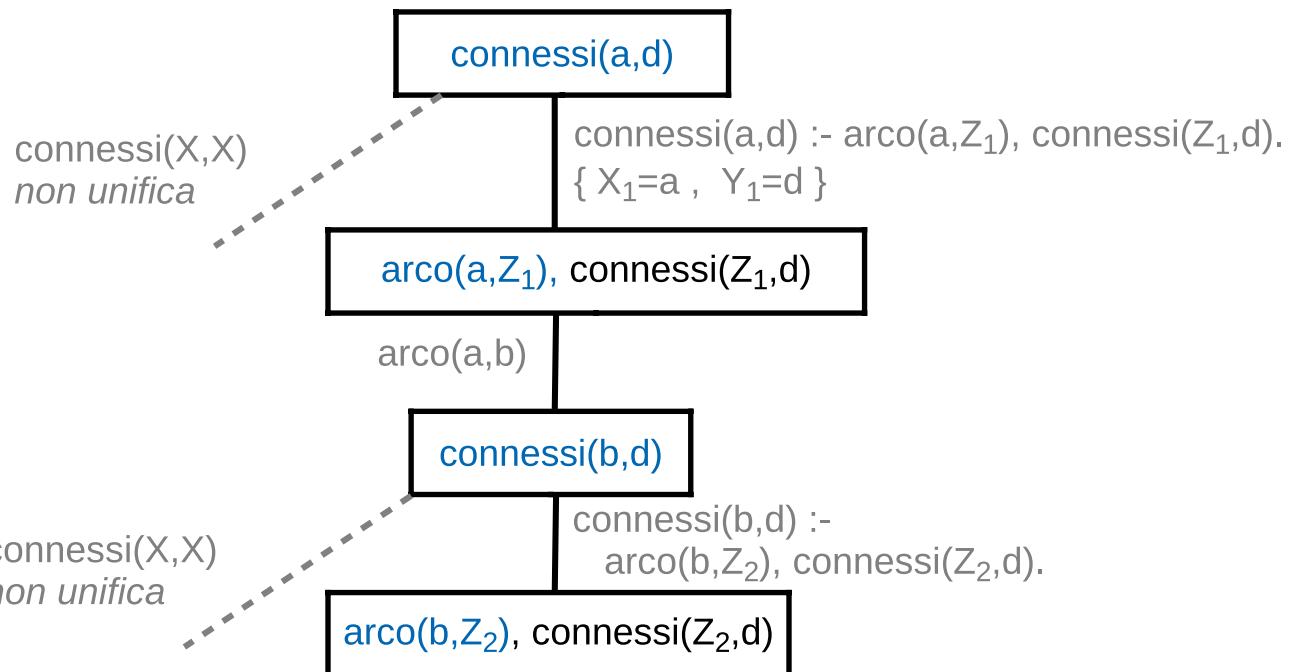


Figure 2.4 A simple graph





Search tree parziale per la query `connessi(a,d)`

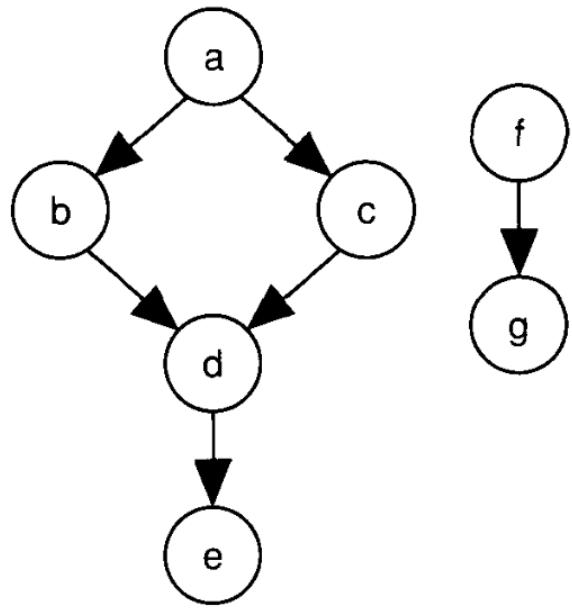
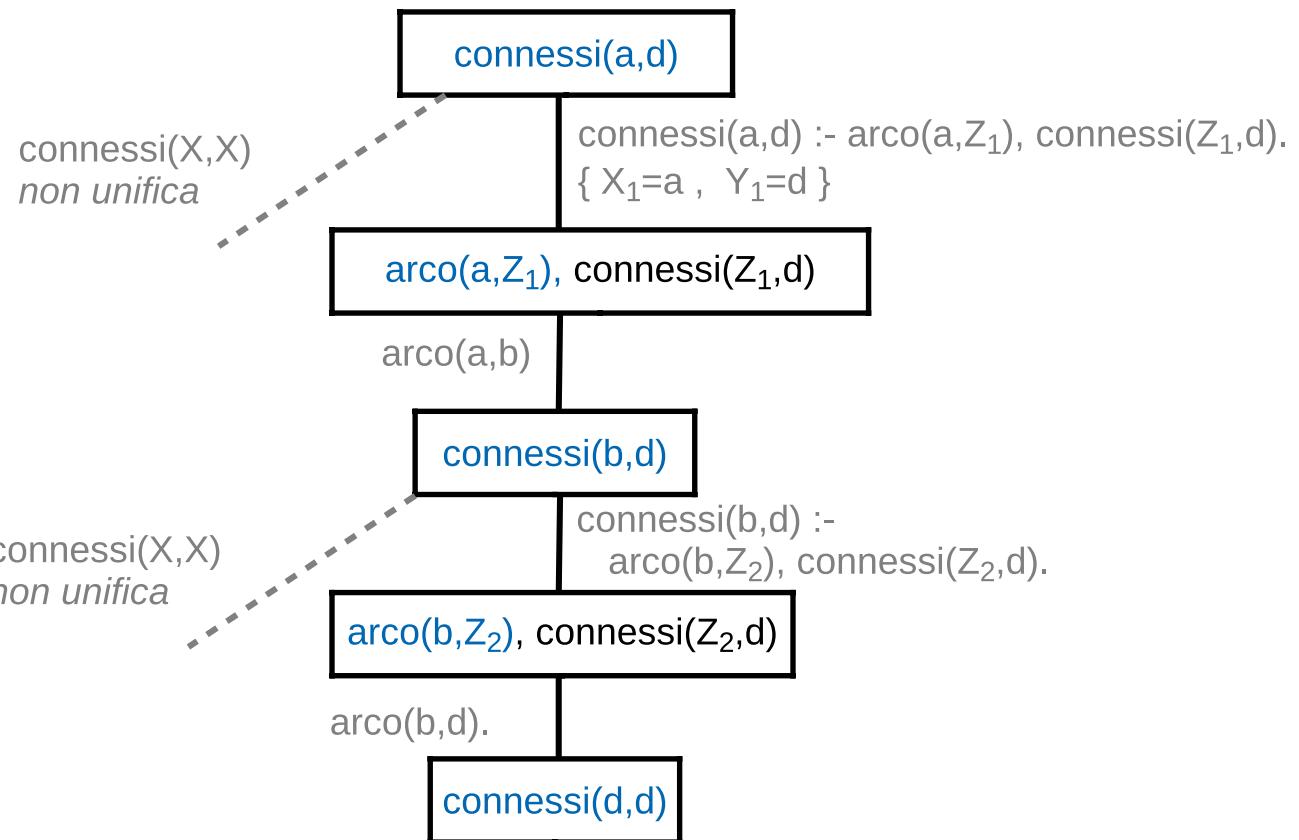


Figure 2.4 A simple graph





Search tree parziale per la query `connessi(a,d)`

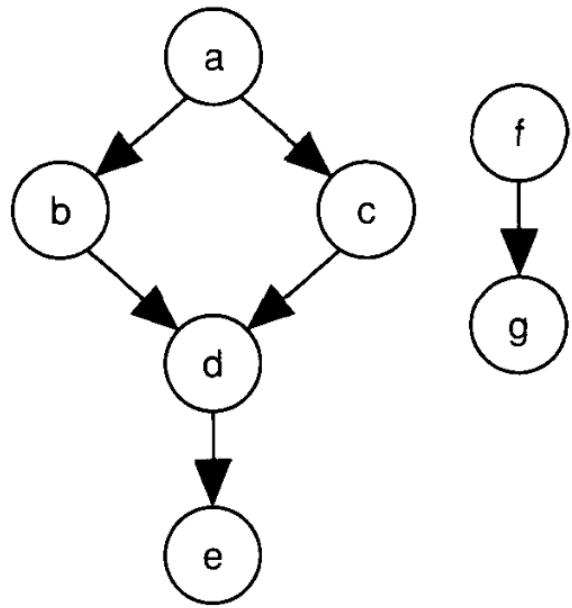
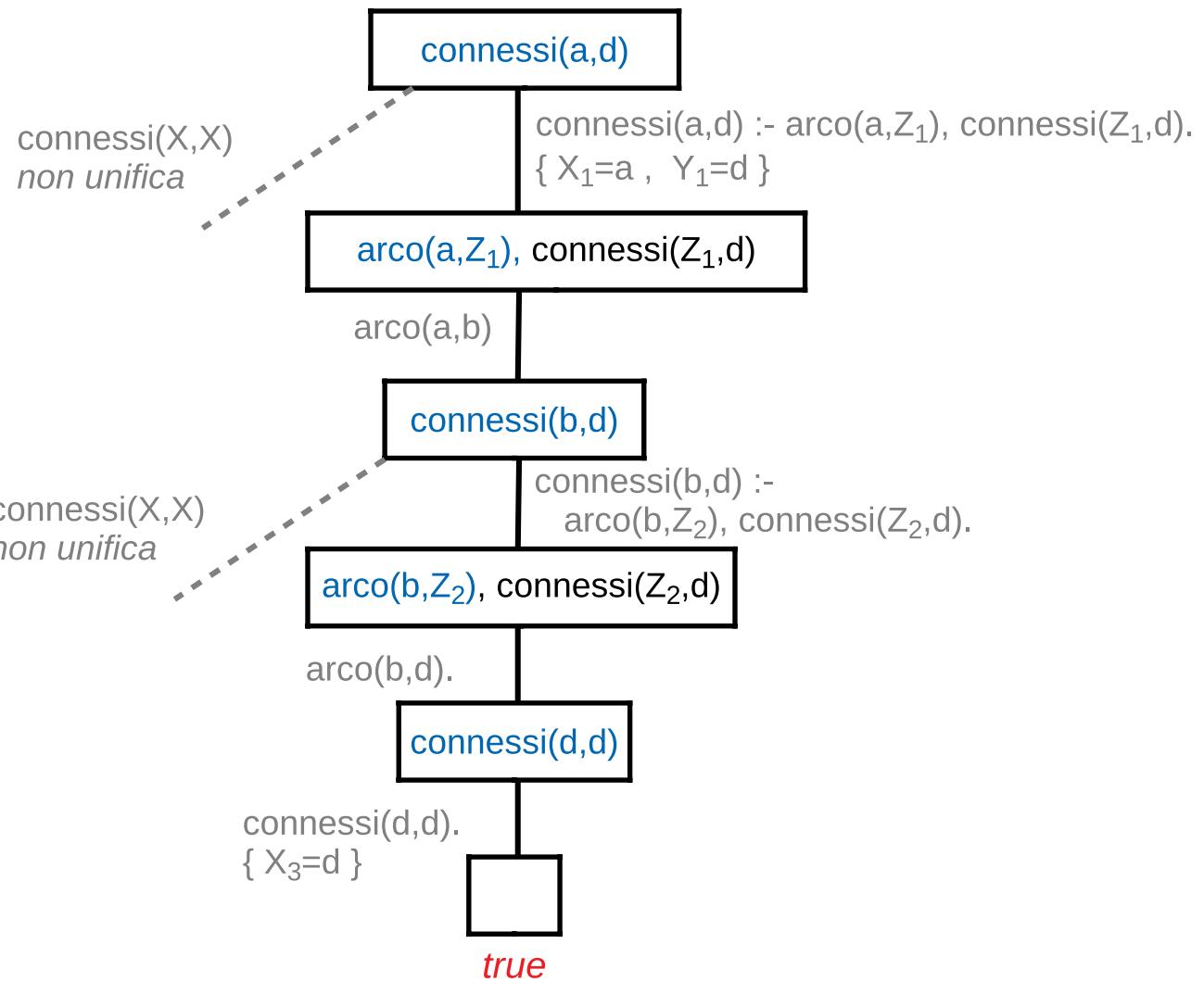


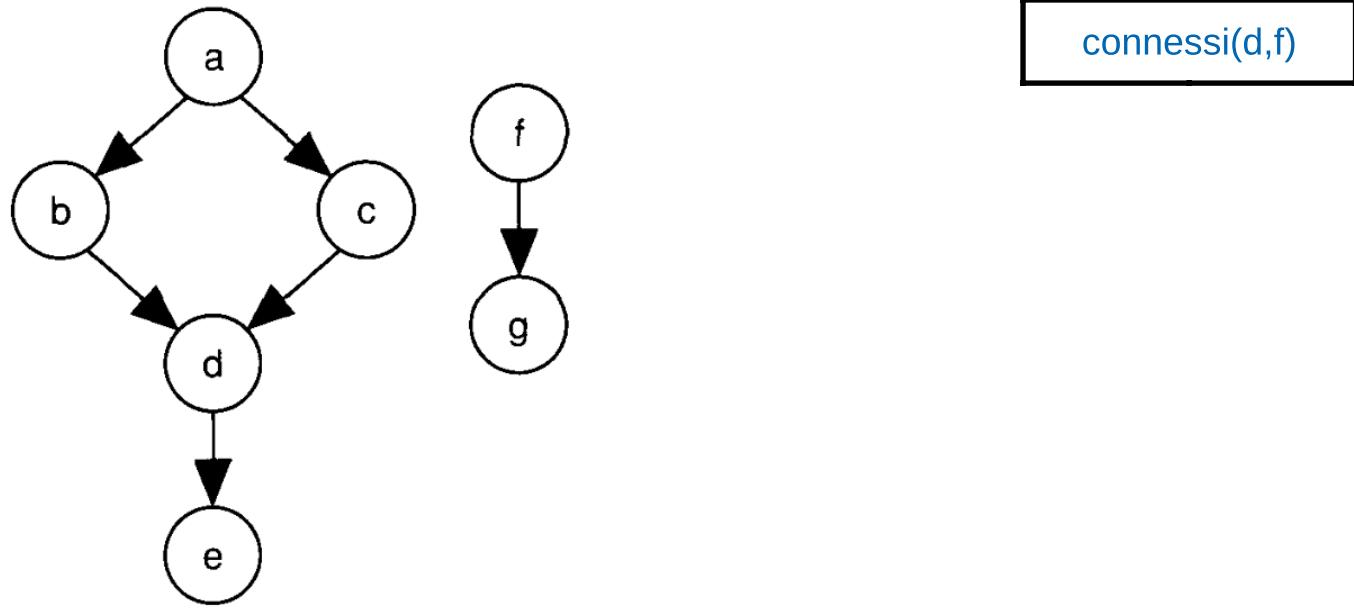
Figure 2.4 A simple graph



Non cerca altre soluzioni perchè la query è ground (inutile restituire altri “true”)



Esempio di search tree per la query $\text{connessi}(d,f)$



connessi(d,f)

Figure 2.4 A simple graph



Esempio di search tree per la query $\text{connessi}(d,f)$

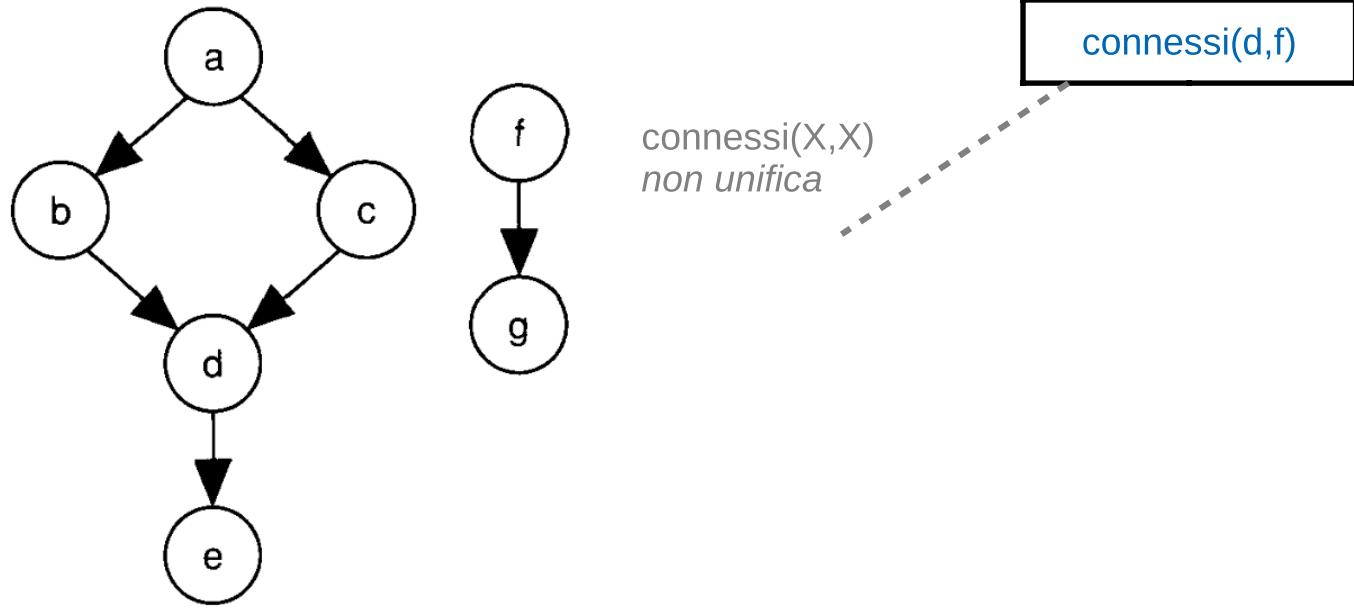
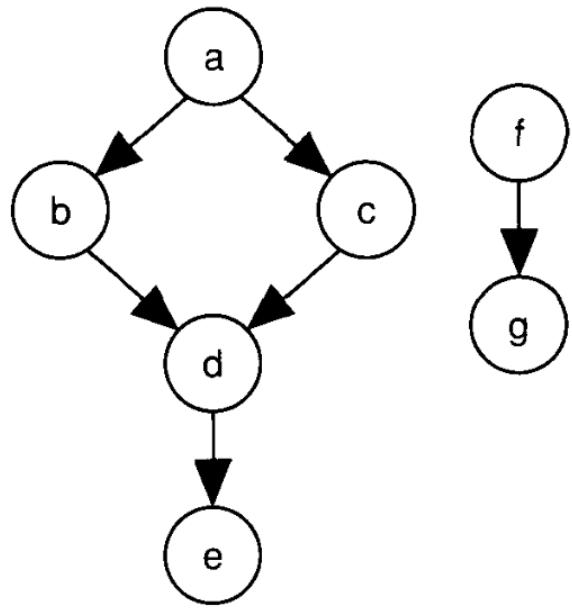


Figure 2.4 A simple graph



Esempio di search tree per la query $\text{connessi}(d,f)$



$\text{connessi}(X,X)$
non unifica

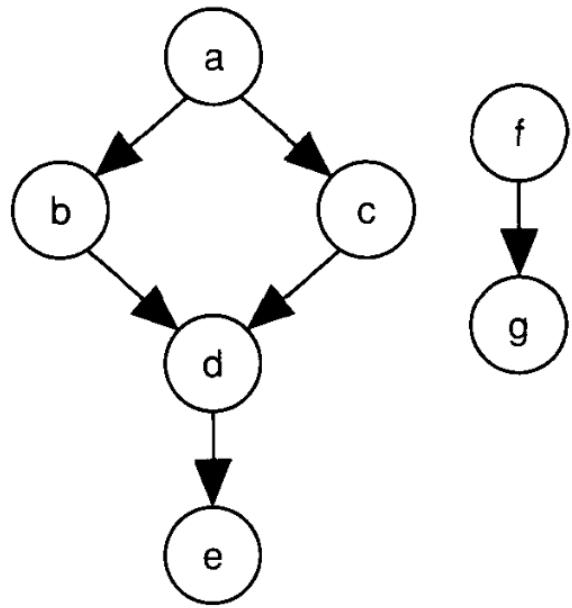
$\text{connessi}(d,f)$

$\text{connessi}(d,f) :- \text{arco}(d,Z_1), \text{connessi}(Z_1,f).$
 $\{ X_1=d , Y_1=f \}$

Figure 2.4 A simple graph



Esempio di search tree per la query $\text{connessi}(d,f)$



$\text{connessi}(X,X)$
non unifica

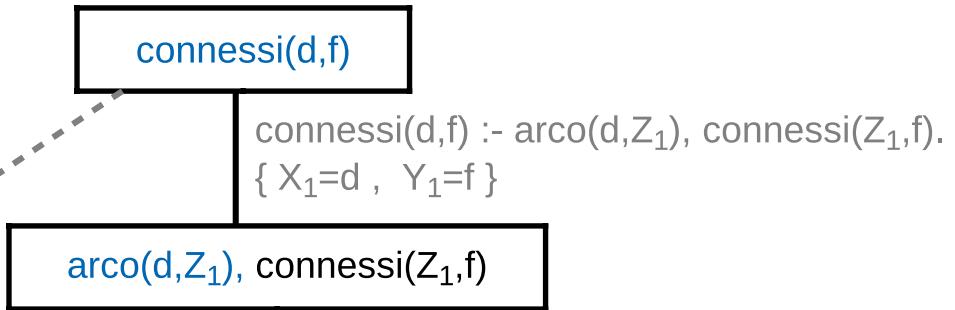
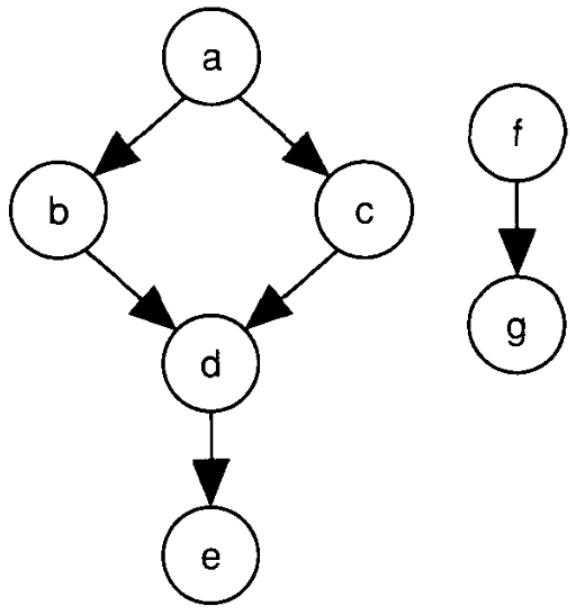


Figure 2.4 A simple graph



Esempio di search tree per la query $\text{connessi}(d,f)$



$\text{connessi}(X,X)$
non unifica

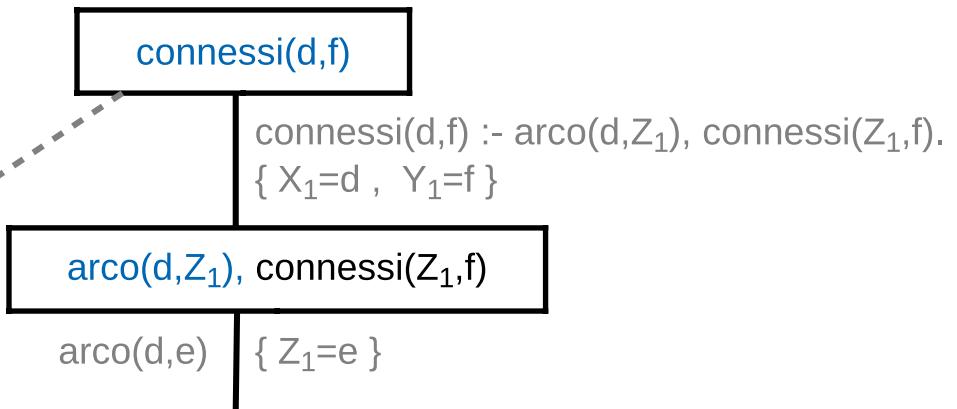
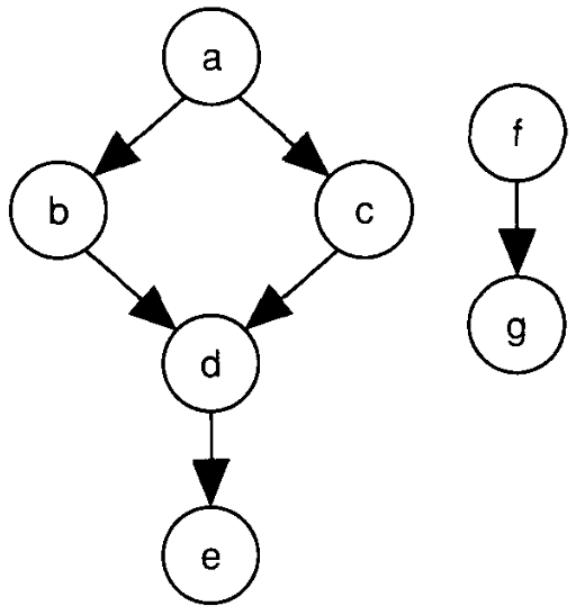


Figure 2.4 A simple graph



Esempio di search tree per la query $\text{connessi}(d,f)$



$\text{connessi}(X,X)$
non unifica

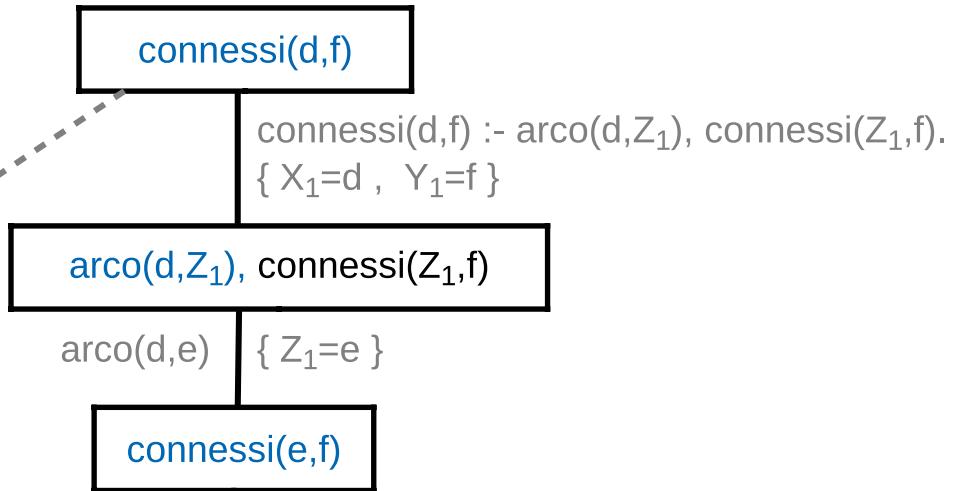
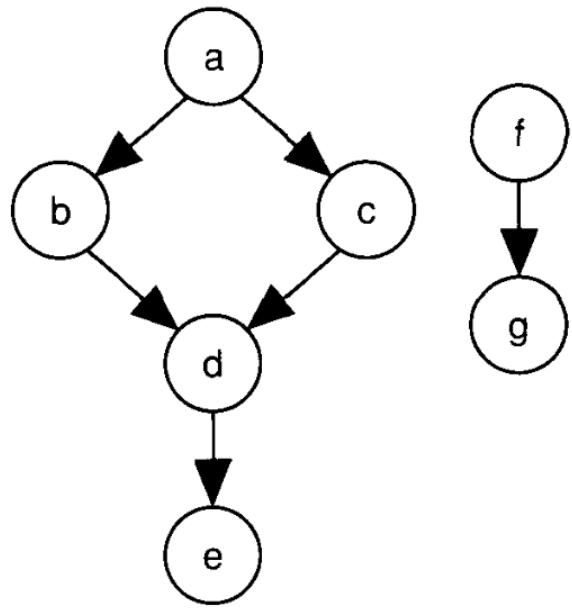


Figure 2.4 A simple graph



Esempio di search tree per la query $\text{connessi}(d,f)$



$\text{connessi}(X,X)$
non unifica

$\text{connessi}(X,X)$
non unifica

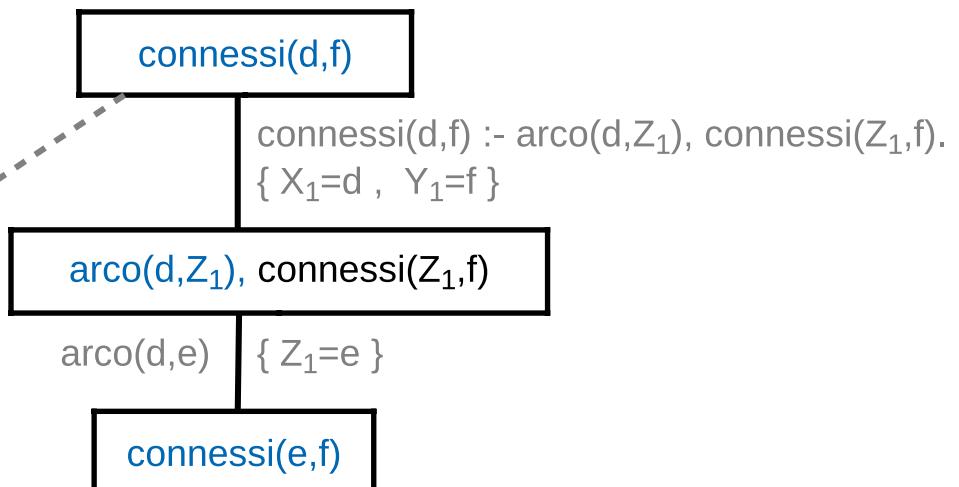
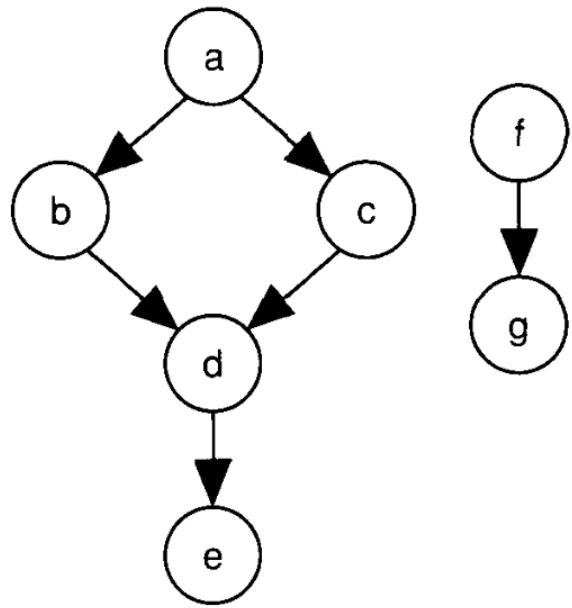


Figure 2.4 A simple graph



Esempio di search tree per la query $\text{connessi}(d,f)$



connessi(X,X)
non unifica

connessi(X,X)
non unifica

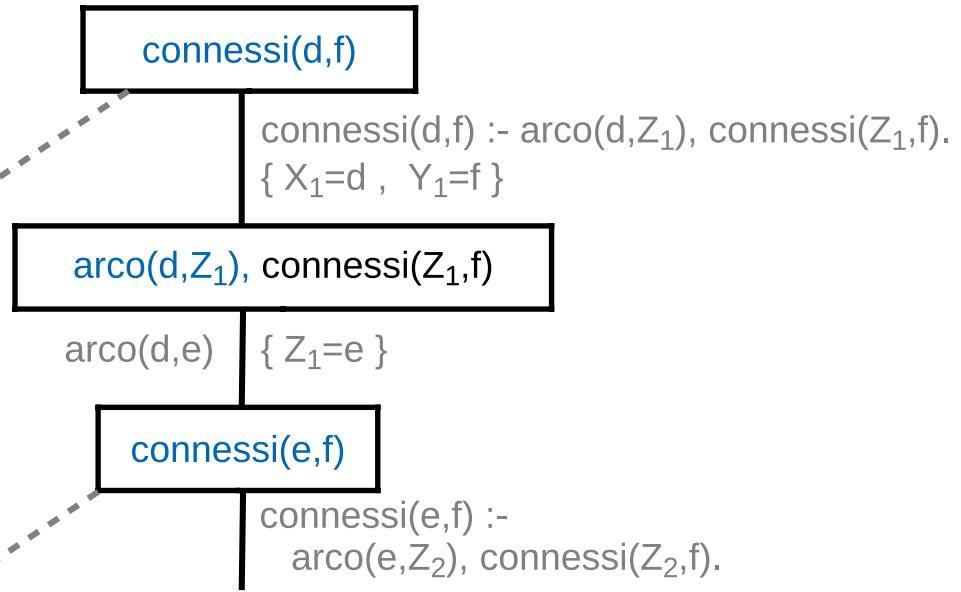


Figure 2.4 A simple graph



Esempio di search tree per la query $\text{connessi}(d,f)$

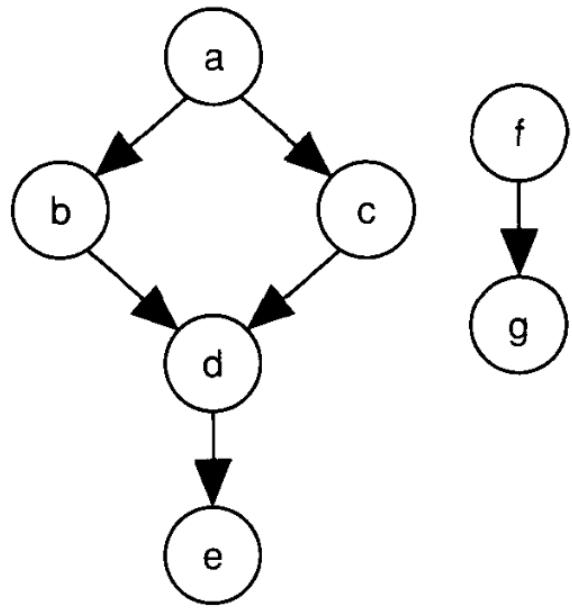
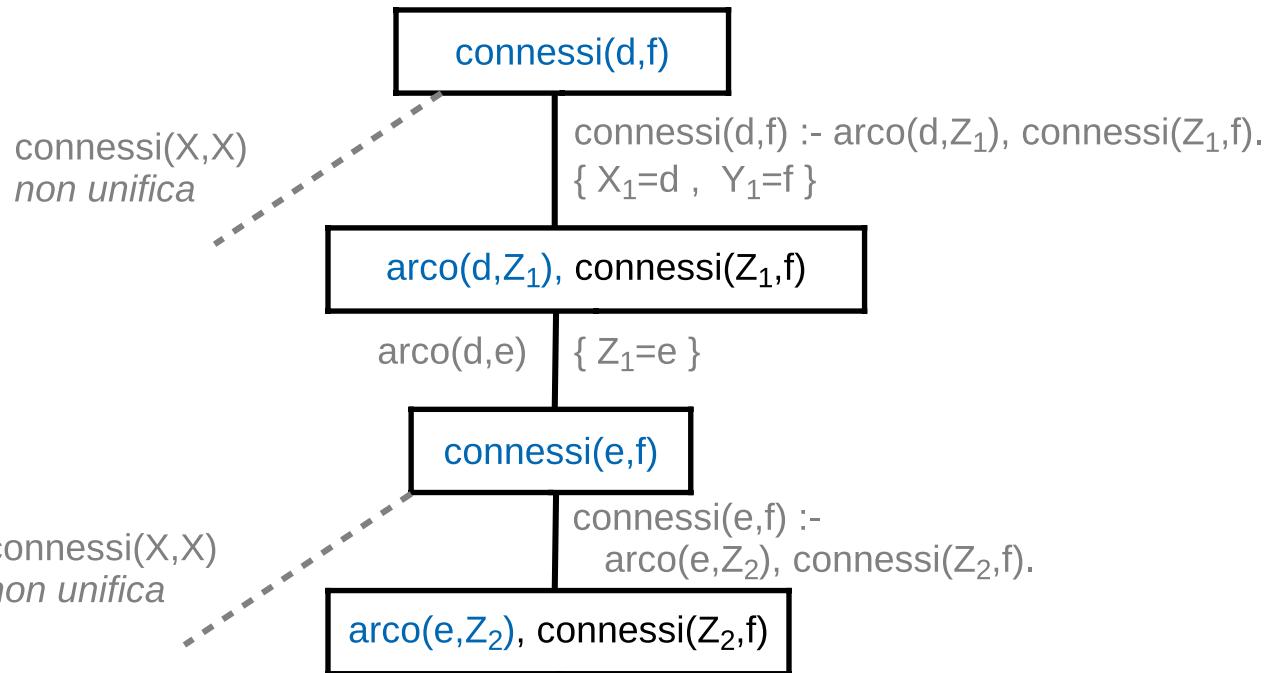


Figure 2.4 A simple graph





Esempio di search tree per la query $\text{connessi}(d,f)$

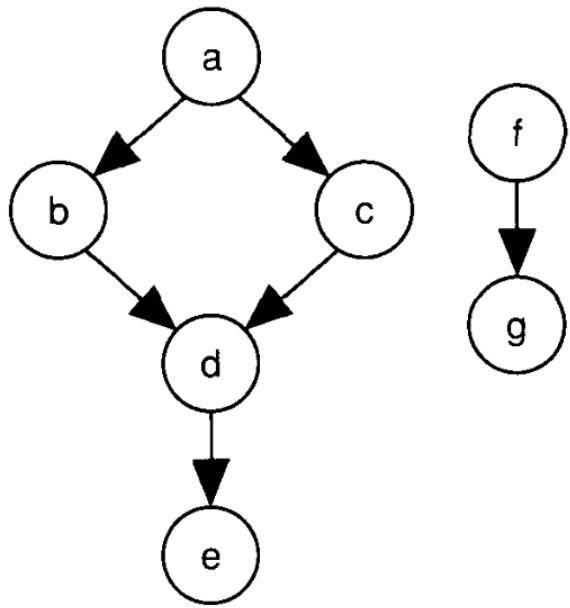
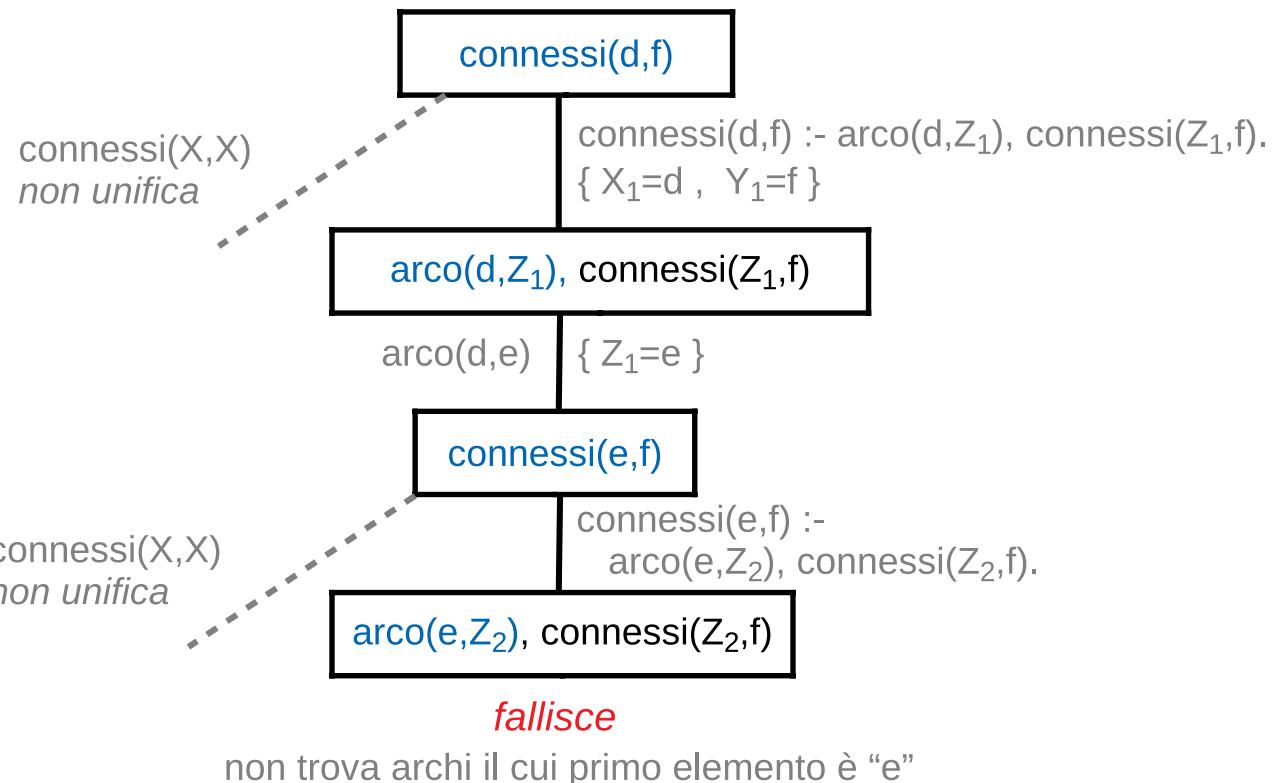


Figure 2.4 A simple graph





Esempio di search tree per la query $\text{connessi}(d,f)$

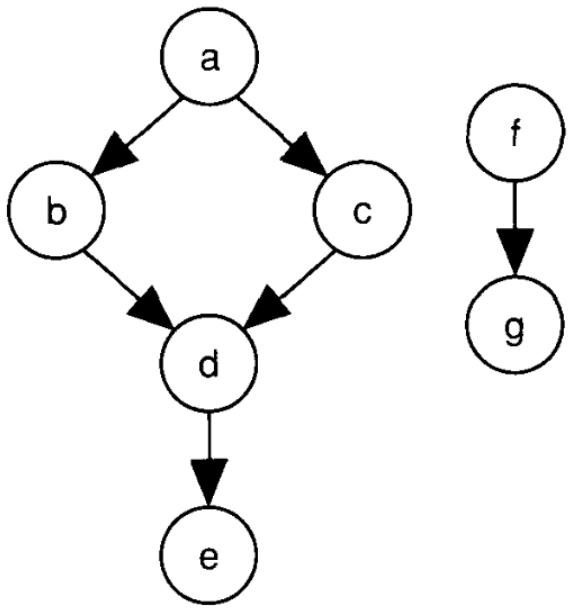
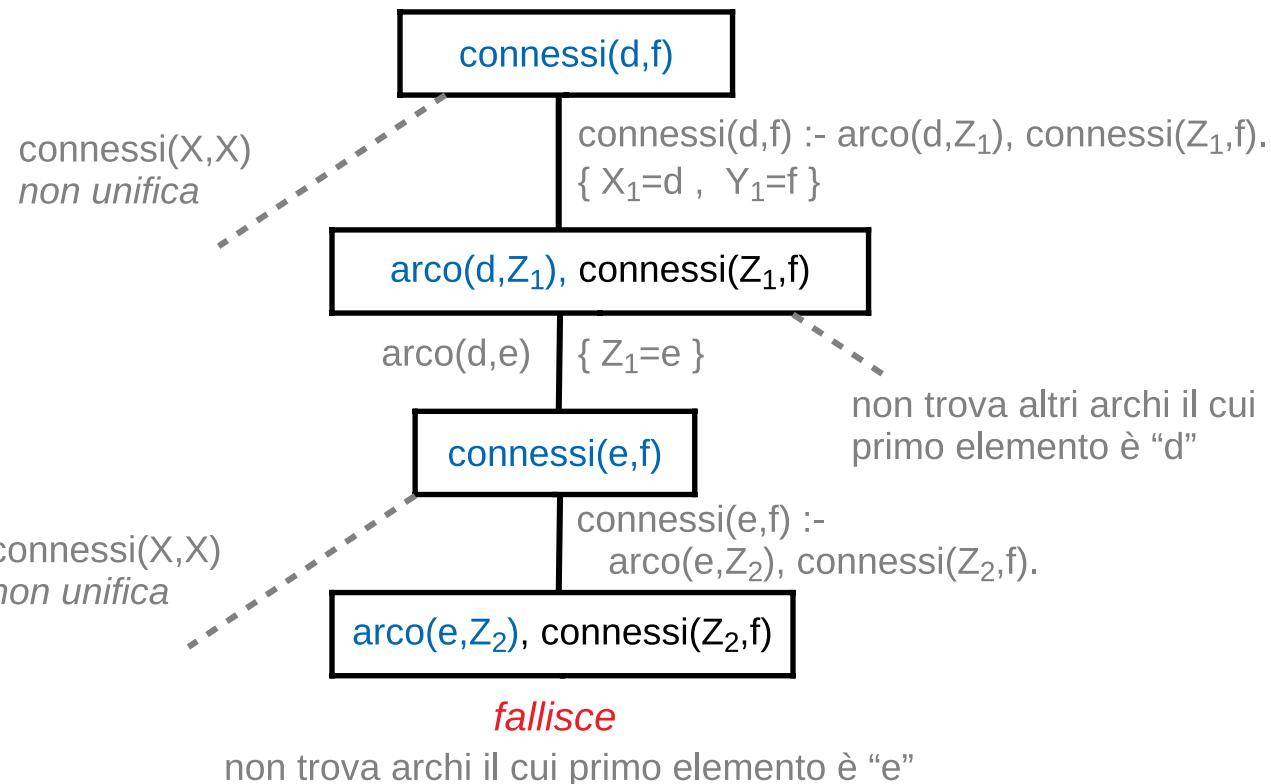


Figure 2.4 A simple graph





Esempio di search tree per la query $\text{connessi}(d,f)$

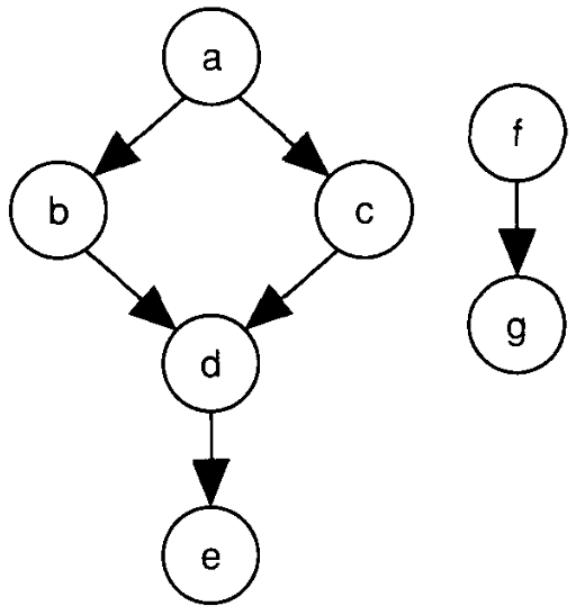
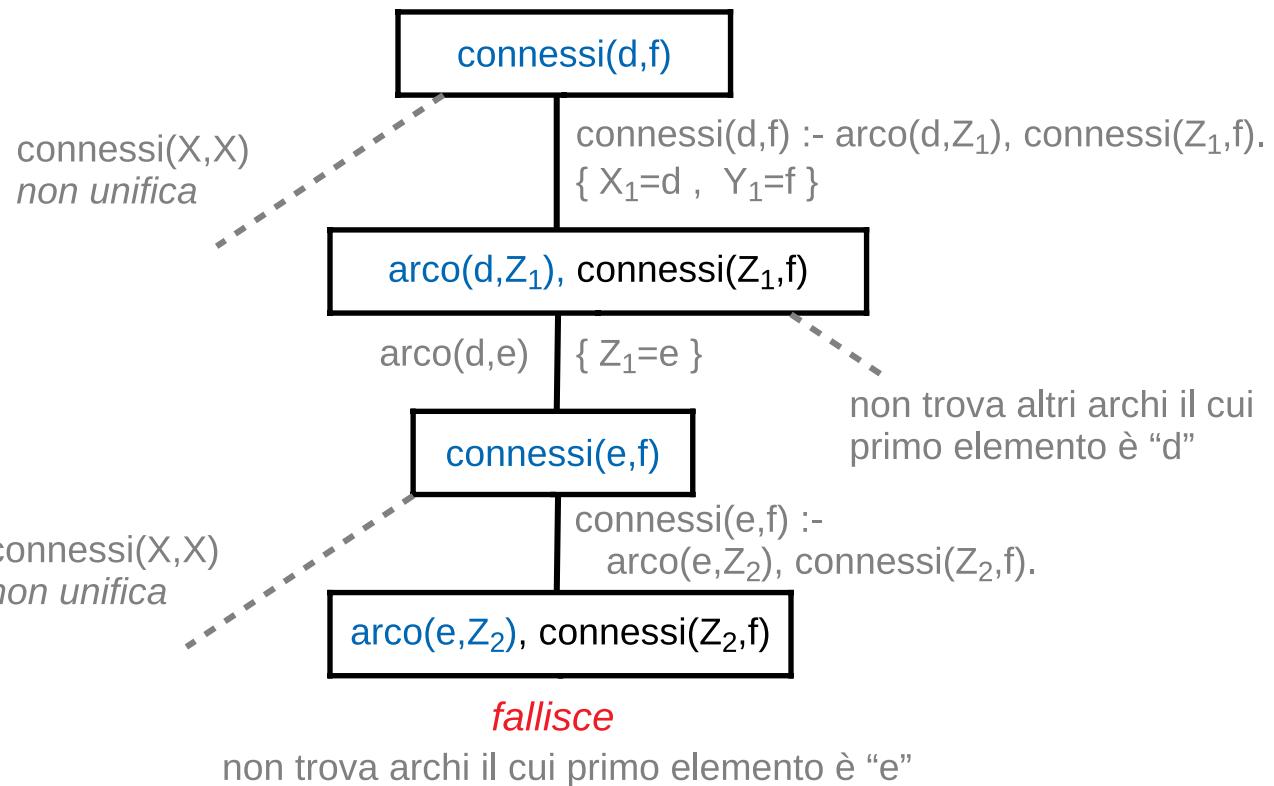


Figure 2.4 A simple graph



La risposta è false



Query non terminanti

Invece delle regole viste prima:

```
connessi(X,X).  
connessi(X,Y) :- arco(X,Z), connessi(Z,Y).
```

...proviamo a usare le seguenti:

```
connessi2(X,X).
```



Query non terminanti

Invece delle regole viste prima:

```
connessi(X,X).  
connessi(X,Y) :- arco(X,Z), connessi(Z,Y).
```

...proviamo a usare le seguenti:

```
connessi2(X,X).  
connessi2(X,Y) :- arco(X,Y).
```



Query non terminanti

Invece delle regole viste prima:

```
connessi(X,X).  
connessi(X,Y) :- arco(X,Z), connessi(Z,Y).
```

...proviamo a usare le seguenti:

```
connessi2(X,X).  
connessi2(X,Y) :- arco(X,Y).  
connessi2(X,Y) :- connessi2(X,Z), connessi2(Z,Y).
```



Query non terminanti

```
connessi2(X,X).  
connessi2(X,Y) :- arco(X,Y).  
connessi2(X,Y) :- connessi2(X,Z), connessi2(Z,Y).
```

Query (vera): `connessi2(a,d)`

Risposta:

```
ERROR: Stack limit (1.0Gb) exceeded  
...  
ERROR: Probable infinite recursion (cycle):  
ERROR: [4,171,061] user:connessi2(a, d)  
ERROR: [4,171,060] user:connessi2(a, d)
```

Esercizio: spiegare l'errore sviluppando il search tree

Conclusione: la definizione di `connessi2` è logicamente corretta, ma non adatta allo schema di valutazione Prolog



Query non terminanti

Attenzione: anche connessi può non terminare, se il grafo contiene cicli!

Ad esempio, aggiungiamo un ciclo al grafo precedente:

```
arco(d,a).
```

Query (falsa): `connessi(a,f)`

Risposta:

```
ERROR: Stack limit (1.0Gb) exceeded  
...
```

Per risolvere questo problema, ci serviranno nuovi costrutti (negazione e liste)



Esempio simile: gli antenati

- Aggiungiamo ora al programma sulla genealogia biblica la nozione di *antenato* (*ancestor*)
- Gli antenati di X sono i suoi genitori, nonni, bisnonni, ...

Paradigma logico

Prolog

Costrutti Prolog

Unificazione

Conjunctive queries

Le Regole

Ragionamento

Derivazioni

Overloading

Wildcards

Analisi di circuiti

Cammini su grafi

Cammini su grafi

Cammini su grafi

Cammini su grafi

Relational algebra

Negazione

Liste

Applicazioni

Programmazione
nondeterministica

Unicità di Prolog



Esempio simile: gli antenati

- Aggiungiamo ora al programma sulla genealogia biblica la nozione di *antenato* (*ancestor*)
- Gli antenati di X sono i suoi genitori, nonni, bisnonni, ...
- Definizione ricorsiva: X è un antenato di Y **se** vale una di queste condizioni:
 1. X è genitore di Y (caso base)
 2. X è genitore di Z e Z è antenato di Y (per qualche Z)

```
ancestor(X,Y) :- parent(X,Y).  
ancestor(X,Y) :- parent(X,Z), ancestor(Z,Y).
```

[Paradigma logico](#)

[Prolog](#)

[Costrutti Prolog](#)

[Unificazione](#)

[Conjunctive queries](#)

[Le Regole](#)

[Ragionamento](#)

[Derivazioni](#)

[Overloading](#)

[Wildcards](#)

[Analisi di circuiti](#)

[Cammini su grafi](#)

[Cammini su grafi](#)

[Cammini su grafi](#)

[Relational algebra](#)

[Negazione](#)

[Liste](#)

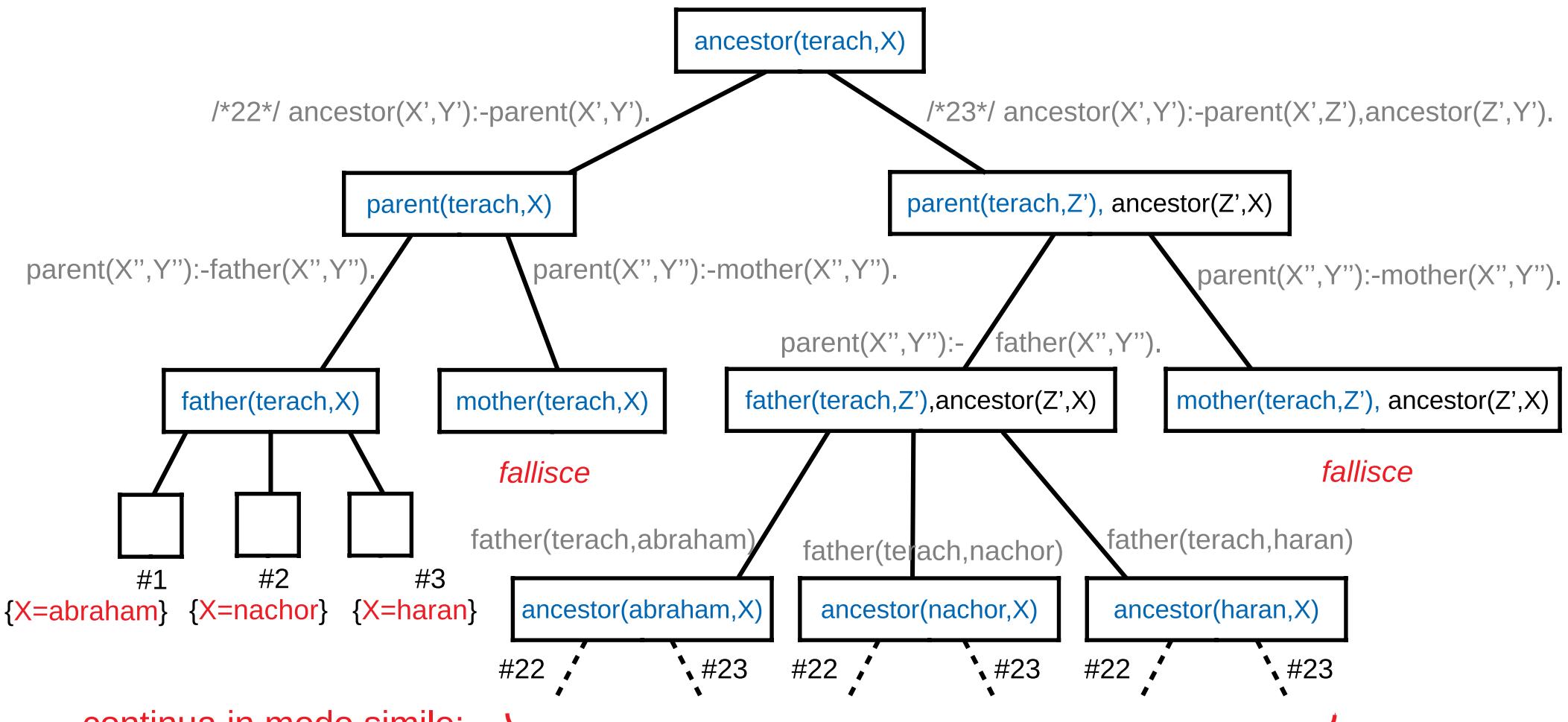
[Applicazioni](#)

[Programmazione nondeterministica](#)

[Unicità di Prolog](#)



Esempio di search tree per ancestor



continua in modo simile:
lega X ai discendenti di
abraham, nachor e haran



Esercizio

[Paradigma logico](#)

[Prolog](#)

[Costrutti Prolog](#)

[Unificazione](#)

[Conjunctive queries](#)

[Le Regole](#)

[Ragionamento](#)

[Derivazioni](#)

[Overloading](#)

[Wildcards](#)

[Analisi di circuiti](#)

[Cammini su grafi](#)

[Cammini su grafi](#)

[Cammini su grafi](#)

[Cammini su grafi](#)

[Relational algebra](#)

[Negazione](#)

[Liste](#)

[Applicazioni](#)

[Programmazione nondeterministica](#)

[Unicità di Prolog](#)

Scrivere un programma Prolog che rappresenta le relazioni di assegnabilità e sottotipo in Java. Il programma deve supportare i seguenti tipi:

- Tutti i tipi primitivi
- I tipi object, string, null
- I tipi person, employee, manager, ciascuno sottotipo del precedente
- I tipi array, tramite un funtore array

Esempi di query:

1. `assegnabile(int, float).` → true
2. `assegnabile(float, int).` → false
3. `sottotipo(int, float).` → false
4. `sottotipo(employee, object).` → true
5. `sottotipo(employee, employee).` → true
6. `sottotipo(array(int), object).` → true
7. `assegnabile(array(employee), array(person)).` → true



Prolog e l'algebra relazionale

- Ogni tabella relazionale r si può rappresentare con dei fatti

$r :$

a_1	b_1	c_1	d_1
a_2	b_2	c_2	d_2
a_3	b_3	c_3	d_3
:	:	:	:
:	:	:	:

```
r(a1, b1, c1, d1) .  
r(a2, b2, c2, d2) .  
r(a3, b3, c3, d3) .  
.  
.  
.
```

Paradigma logico

Prolog

Costrutti Prolog

Unificazione

Conjunctive queries

Le Regole

Ragionamento

Derivazioni

Overloading

Wildcards

Analisi di circuiti

Cammini su grafi

Cammini su grafi

Cammini su grafi

Cammini su grafi

Relational algebra

Negazione

Liste

Applicazioni

Programmazione nondeterministica

Unicità di Prolog



Prolog e l'algebra relazionale

- Ogni tabella relazionale r si può rappresentare con dei fatti

$r :$

a_1	b_1	c_1	d_1
a_2	b_2	c_2	d_2
a_3	b_3	c_3	d_3
:	:	:	:
:	:	:	:

```
r(a1, b1, c1, d1) .  
r(a2, b2, c2, d2) .  
r(a3, b3, c3, d3) .  
.  
.  
.
```

- Con le regole si può facilmente simulare l'algebra relazionale (ovvero le query SQL)
 - ◆ Prolog genera le n-uple della risposta una per una

Paradigma logico

Prolog

Costrutti Prolog

Unificazione

Conjunctive queries

Le Regole

Ragionamento

Derivazioni

Overloading

Wildcards

Analisi di circuiti

Cammini su grafi

Cammini su grafi

Cammini su grafi

Cammini su grafi

Relational algebra

Negazione

Liste

Applicazioni

Programmazione nondeterministica

Unicità di Prolog



Prolog e l'algebra relazionale

[Paradigma logico](#)

[Prolog](#)

[Costrutti Prolog](#)

[Unificazione](#)

[Conjunctive queries](#)

[Le Regole](#)

[Ragionamento](#)

[Derivazioni](#)

[Overloading](#)

[Wildcards](#)

[Analisi di circuiti](#)

[Cammini su grafi](#)

[Cammini su grafi](#)

[Cammini su grafi](#)

[Cammini su grafi](#)

[Relational algebra](#)

[Negazione](#)

[Liste](#)

[Applicazioni](#)

[Programmazione
nondeterministica](#)

[Unicità di Prolog](#)

```
/* UNIONE di r e s */  
r_union_s(X1,...,Xn) :- r(X1,...,Xn).  
r_union_s(X1,...,Xn) :- s(X1,...,Xn).
```



Prolog e l'algebra relazionale

[Paradigma logico](#)

[Prolog](#)

[Costrutti Prolog](#)

[Unificazione](#)

[Conjunctive queries](#)

[Le Regole](#)

[Ragionamento](#)

Derivazioni

Overloading

Wildcards

Analisi di circuiti

Cammini su grafi

Cammini su grafi

Cammini su grafi

Cammini su grafi

Relational algebra

Negazione

[Liste](#)

[Applicazioni](#)

Programmazione
nondeterministica

Unicità di Prolog

```
/* UNIONE di r e s */  
r_union_s(X1,...,Xn) :- r(X1,...,Xn).  
r_union_s(X1,...,Xn) :- s(X1,...,Xn).  
  
/* INTERSEZIONE di r e s */  
r_inters_s(X1,...,Xn) :- r(X1,...,Xn), s(X1,...,Xn).
```



Prolog e l'algebra relazionale

[Paradigma logico](#)

[Prolog](#)

[Costrutti Prolog](#)

[Unificazione](#)

[Conjunctive queries](#)

[Le Regole](#)

[Ragionamento](#)

[Derivazioni](#)

[Overloading](#)

[Wildcards](#)

[Analisi di circuiti](#)

[Cammini su grafi](#)

[Cammini su grafi](#)

[Cammini su grafi](#)

[Cammini su grafi](#)

[Relational algebra](#)

[Negazione](#)

[Liste](#)

[Applicazioni](#)

[Programmazione nondeterministica](#)

[Unicità di Prolog](#)

```
/* UNIONE di r e s */
r_union_s(X1,...,Xn) :- r(X1,...,Xn).
r_union_s(X1,...,Xn) :- s(X1,...,Xn).

/* INTERSEZIONE di r e s */
r_inters_s(X1,...,Xn) :- r(X1,...,Xn), s(X1,...,Xn).

/* PROIEZIONE di r, ad es. su colonne 1 e 3 */
r13(X1,X3) :- r(X1,...,Xn).
```



Prolog e l'algebra relazionale

[Paradigma logico](#)

[Prolog](#)

[Costrutti Prolog](#)

[Unificazione](#)

[Conjunctive queries](#)

[Le Regole](#)

[Ragionamento](#)

[Derivazioni](#)

[Overloading](#)

[Wildcards](#)

[Analisi di circuiti](#)

[Cammini su grafi](#)

[Cammini su grafi](#)

[Cammini su grafi](#)

[Relational algebra](#)

[Negazione](#)

[Liste](#)

[Applicazioni](#)

[Programmazione
nondeterministica](#)

[Unicità di Prolog](#)

```
/* UNIONE di r e s */
r_union_s(X1,...,Xn) :- r(X1,...,Xn).
r_union_s(X1,...,Xn) :- s(X1,...,Xn).

/* INTERSEZIONE di r e s */
r_inters_s(X1,...,Xn) :- r(X1,...,Xn), s(X1,...,Xn).

/* PROIEZIONE di r, ad es. su colonne 1 e 3 */
r13(X1,X3) :- r(X1,...,Xn).

/* SELEZIONE di r */
r_sel(X1,...,Xn) :- r(X1,...,Xn), <condizione>.

/* ad esempio: */
r_with_2_less_than_3(X1,...,Xn) :- r(X1,...,Xn), X2 < X3.
```



Prolog e l'algebra relazionale

Paradigma logico

Prolog

Costrutti Prolog

Unificazione

Conjunctive queries

Le Regole

Ragionamento

Derivazioni

Overloading

Wildcards

Analisi di circuiti

Cammini su grafi

Cammini su grafi

Cammini su grafi

Cammini su grafi

Relational algebra

Negazione

Liste

Applicazioni

Programmazione nondeterministica

Unicità di Prolog

```
/* UNIONE di r e s */
r_union_s(X1,...,Xn) :- r(X1,...,Xn).
r_union_s(X1,...,Xn) :- s(X1,...,Xn).

/* INTERSEZIONE di r e s */
r_inters_s(X1,...,Xn) :- r(X1,...,Xn), s(X1,...,Xn).

/* PROIEZIONE di r, ad es. su colonne 1 e 3 */
r13(X1,X3) :- r(X1,...,Xn).

/* SELEZIONE di r */
r_sel(X1,...,Xn) :- r(X1,...,Xn), <condizione>.

/* ad esempio: */
r_with_2_less_than_3(X1,...,Xn) :- r(X1,...,Xn), X2 < X3.
```

- Scrivere prodotto cartesiano e join su colonne 1 e 2 come esercizio



Prolog e l'algebra relazionale

[Paradigma logico](#)

[Prolog](#)

[Costrutti Prolog](#)

[Unificazione](#)

[Conjunctive queries](#)

[Le Regole](#)

[Ragionamento](#)

[Derivazioni](#)

[Overloading](#)

[Wildcards](#)

[Analisi di circuiti](#)

[Cammini su grafi](#)

[Cammini su grafi](#)

[Cammini su grafi](#)

[Cammini su grafi](#)

[Relational algebra](#)

[Negazione](#)

[Liste](#)

[Applicazioni](#)

[Programmazione nondeterministica](#)

[Unicità di Prolog](#)

- Per la differenza tra relazioni occorre un operatore di **negazione** denotato da **\+¹**

```
/* DIFFERENZA INSIEMISTICA tra r e s */  
r_minus_s(X1,...,Xn) :- r(X1,...,Xn), \+ s(X1,...,Xn).
```



Prolog e l'algebra relazionale

[Paradigma logico](#)

[Prolog](#)

[Costrutti Prolog](#)

[Unificazione](#)

[Conjunctive queries](#)

[Le Regole](#)

[Ragionamento](#)

[Derivazioni](#)

[Overloading](#)

[Wildcards](#)

[Analisi di circuiti](#)

[Cammini su grafi](#)

[Cammini su grafi](#)

[Cammini su grafi](#)

[Cammini su grafi](#)

[Relational algebra](#)

Negazione

[Liste](#)

[Applicazioni](#)

[Programmazione nondeterministica](#)

[Unicità di Prolog](#)

- Per la differenza tra relazioni occorre un operatore di **negazione** denotato da \textbackslash+ ¹

```
/* DIFFERENZA INSIEMISTICA tra r e s */  
r_minus_s(X1,...,Xn) :- r(X1,...,Xn), \+ s(X1,...,Xn).
```

- La negazione trasforma false in true, ovvero fallimenti in successi.
 - ◆ $\text{\textbackslash+ p}(X_1, \dots, X_n)$ è *true* se tutti i rami del suo albero di ricerca terminano con un fallimento



Prolog e l'algebra relazionale

Paradigma logico

Prolog

Costrutti Prolog

Unificazione

Conjunctive queries

Le Regole

Ragionamento

Derivazioni

Overloading

Wildcards

Analisi di circuiti

Cammini su grafi

Cammini su grafi

Cammini su grafi

Cammini su grafi

Relational algebra

Negazione

Liste

Applicazioni

Programmazione nondeterministica

Unicità di Prolog

- Per la differenza tra relazioni occorre un operatore di **negazione** denotato da **\+¹**

```
/* DIFFERENZA INSIEMISTICA tra r e s */  
r_minus_s(X1,...,Xn) :- r(X1,...,Xn), \+ s(X1,...,Xn).
```

- La negazione trasforma false in true, ovvero fallimenti in successi.
 - ◆ $\text{\+ } p(X_1, \dots, X_n)$ è *true* se tutti i rami del suo albero di ricerca terminano con un fallimento
 - ◆ L'albero deve essere finito. Se un programma cade in una ricorsione infinita ovviamente non termina.

¹Nel libro si usa **not**



[Paradigma logico](#)

[Prolog](#)

[Costrutti Prolog](#)

[Unificazione](#)

[Conjunctive queries](#)

[Le Regole](#)

[Ragionamento](#)

Liste

Sintassi

Il predicato member

Prolog vs ML

IN e OUT

Append

Negazione

[Applicazioni](#)

Programmazione
nondeterministica

[Unicità di Prolog](#)

Liste



Le liste in Prolog

- La rappresentazione è analoga a quella in ML
 - ◆ costruttore lista vuota: []
 - ◆ costruttore nodi: [elem|resto]

[Paradigma logico](#)

[Prolog](#)

[Costrutti Prolog](#)

[Unificazione](#)

[Conjunctive queries](#)

[Le Regole](#)

[Ragionamento](#)

[Liste](#)

Sintassi

Il predicato member

Prolog vs ML

IN e OUT

Append

Negazione

[Applicazioni](#)

Programmazione
nondeterministica

[Unicità di Prolog](#)



Le liste in Prolog

[Paradigma logico](#)

[Prolog](#)

[Costrutti Prolog](#)

[Unificazione](#)

[Conjunctive queries](#)

[Le Regole](#)

[Ragionamento](#)

[Liste](#)

Sintassi

Il predicato member

Prolog vs ML

IN e OUT

Append

Negazione

[Applicazioni](#)

Programmazione
nondeterministica

[Unicità di Prolog](#)

■ La rappresentazione è analoga a quella in ML

- ◆ costruttore lista vuota: []
- ◆ costruttore nodi: [elem|resto]

■ Notazioni alternative equivalenti:

Abbreviata
[a]

Costruttori esplicativi
[a | []]



Le liste in Prolog

[Paradigma logico](#)

[Prolog](#)

[Costrutti Prolog](#)

[Unificazione](#)

[Conjunctive queries](#)

[Le Regole](#)

[Ragionamento](#)

[Liste](#)

[Sintassi](#)

[Il predicato member](#)

[Prolog vs ML](#)

[IN e OUT](#)

[Append](#)

[Negazione](#)

[Applicazioni](#)

[Programmazione
nondeterministica](#)

[Unicità di Prolog](#)

■ La rappresentazione è analoga a quella in ML

- ◆ costruttore lista vuota: []
- ◆ costruttore nodi: [elem|resto]

■ Notazioni alternative equivalenti:

Abbreviata

[a]

[a , b]

Costruttori esplicativi

[a | []]

[a | [b | []]]



Le liste in Prolog

[Paradigma logico](#)

[Prolog](#)

[Costrutti Prolog](#)

[Unificazione](#)

[Conjunctive queries](#)

[Le Regole](#)

[Ragionamento](#)

[Liste](#)

Sintassi

Il predicato member

Prolog vs ML

IN e OUT

Append

Negazione

[Applicazioni](#)

[Programmazione
nondeterministica](#)

[Unicità di Prolog](#)

■ La rappresentazione è analoga a quella in ML

- ◆ costruttore lista vuota: []
- ◆ costruttore nodi: [elem|resto]

■ Notazioni alternative equivalenti:

Abbreviata

[a]
[a, b]
[a, b, c]

Costruttori esplicativi

[a | []]
[a | [b | []]]
[a | [b | [c | []]]]



Le liste in Prolog

[Paradigma logico](#)

[Prolog](#)

[Costrutti Prolog](#)

[Unificazione](#)

[Conjunctive queries](#)

[Le Regole](#)

[Ragionamento](#)

[Liste](#)

Sintassi

Il predicato member

Prolog vs ML

IN e OUT

Append

Negazione

[Applicazioni](#)

Programmazione
nondeterministica

Unicità di Prolog

■ La rappresentazione è analoga a quella in ML

- ◆ costruttore lista vuota: []
- ◆ costruttore nodi: [elem|resto]

■ Notazioni alternative equivalenti:

Abbreviata	Costruttori espliciti
[a]	[a []]
[a, b]	[a [b []]]
[a, b, c]	[a [b [c []]]]
[a X]	[a X]
[a, b X]	[a [b X]]]



Le liste in Prolog

[Paradigma logico](#)

[Prolog](#)

[Costrutti Prolog](#)

[Unificazione](#)

[Conjunctive queries](#)

[Le Regole](#)

[Ragionamento](#)

[Liste](#)

Sintassi

Il predicato member

Prolog vs ML

IN e OUT

Append

Negazione

[Applicazioni](#)

Programmazione
nondeterministica

[Unicità di Prolog](#)

■ La rappresentazione è analoga a quella in ML

- ◆ costruttore lista vuota: []
- ◆ costruttore nodi: [elem|resto]

■ Notazioni alternative equivalenti:

Abbreviata

[a]
[a, b]
[a, b, c]
[a | X]
[a, b | X]
[a, b, c | X]

Costruttori esplicativi

[a | []]
[a | [b | []]]
[a | [b | [c | []]]]
[a | X]
[a | [b | X]]
[a | [b | [c | X]]]



Le liste in Prolog

[Paradigma logico](#)

[Prolog](#)

[Costrutti Prolog](#)

[Unificazione](#)

[Conjunctive queries](#)

[Le Regole](#)

[Ragionamento](#)

[Liste](#)

Sintassi

Il predicato member

Prolog vs ML

IN e OUT

Append

Negazione

[Applicazioni](#)

Programmazione
nondeterministica

[Unicità di Prolog](#)

■ La rappresentazione è analoga a quella in ML

- ◆ costruttore lista vuota: []
- ◆ costruttore nodi: [elem|resto]

■ Notazioni alternative equivalenti:

Abbreviata

[a]
[a, b]
[a, b, c]
[a|X]
[a, b|X]
[a, b, c|X]

Costruttori esplicativi

[a | []]
[a | [b | []]]
[a | [b | [c | []]]]
[a | X]
[a | [b | X]]
[a | [b | [c | X]]]



Le liste in Prolog

[Paradigma logico](#)

[Prolog](#)

[Costrutti Prolog](#)

[Unificazione](#)

[Conjunctive queries](#)

[Le Regole](#)

[Ragionamento](#)

[Liste](#)

Sintassi

Il predicato member

Prolog vs ML

IN e OUT

Append

Negazione

[Applicazioni](#)

Programmazione
nondeterministica

Unicità di Prolog

■ La rappresentazione è analoga a quella in ML

- ◆ costruttore lista vuota: []
- ◆ costruttore nodi: [elem|resto]

■ Notazioni alternative equivalenti:

Abbreviata

[a]
[a, b]
[a, b, c]
[a | X]
[a, b | X]
[a, b, c | X]

Costruttori esplicativi

[a | []]
[a | [b | []]]
[a | [b | [c | []]]]
[a | X]
[a | [b | X]]
[a | [b | [c | X]]]

■ Notare la possibilità di esprimere liste *parzialmente specificate*, dove alcuni elementi ed eventualmente la coda sono variabili.

[a , **X** , b | **Y**]



Il predicato member

■ Il predicato member cerca un elemento X in una lista L

- ◆ è ovviamente *ricorsivo*
- ◆ usiamo la wildcard ‘_’

```
member(X , [X | _]).  
member(X , [_ | L]) :- member(X , L).
```

[Paradigma logico](#)

[Prolog](#)

[Costrutti Prolog](#)

[Unificazione](#)

[Conjunctive queries](#)

[Le Regole](#)

[Ragionamento](#)

[Liste](#)

[Sintassi](#)

Il predicato member

[Prolog vs ML](#)

[IN e OUT](#)

[Append](#)

[Negazione](#)

[Applicazioni](#)

[Programmazione
nondeterministica](#)

[Unicità di Prolog](#)



Il predicato member

- Il predicato member cerca un elemento X in una lista L

- ◆ è ovviamente *ricorsivo*
 - ◆ usiamo la wildcard ‘_’

```
member(X , [X | _]).  
member(X , [_ | L]) :- member(X , L).
```

- Somiglia alla definizione per casi in ML

- ◆ ma in Prolog posso usare la stessa variabile più volte
 - ◆ per esprimere pattern dove certi elementi sono uguali

[Paradigma logico](#)

[Prolog](#)

[Costrutti Prolog](#)

[Unificazione](#)

[Conjunctive queries](#)

[Le Regole](#)

[Ragionamento](#)

[Liste](#)

Sintassi

Il predicato member

Prolog vs ML

IN e OUT

Append

Negazione

[Applicazioni](#)

Programmazione
nondeterministica

[Unicità di Prolog](#)



Prolog vs ML

Paradigma logico

Prolog

Costrutti Prolog

Unificazione

Conjunctive queries

Le Regole

Ragionamento

Liste

Sintassi

Il predicato member

Prolog vs ML

IN e OUT

Append

Negazione

Applicazioni

Programmazione
nondeterministica

Unicità di Prolog

In Prolog:

```
member(X,[X|_]).  
member(X,[_|L]) :- member(X,L).
```

Traduzione letterale in ML (errore di compilazione):

```
fun member X X:::_ = true  
| member X _::L = member X L;
```

Versione corretta in ML:

```
fun member X (H::L) = X=H orelse member X L  
| member X [] = false;
```



Esempi di derivazione per member

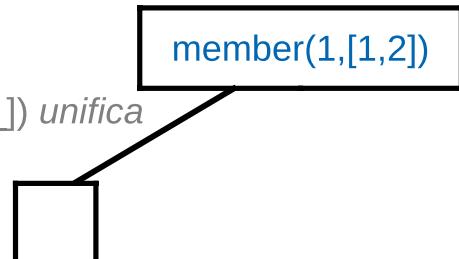
member(1,[1,2])

```
member(X, [X | _]).  
member(X, [_ | L]) :- member(X, L).
```



Esempi di derivazione per member

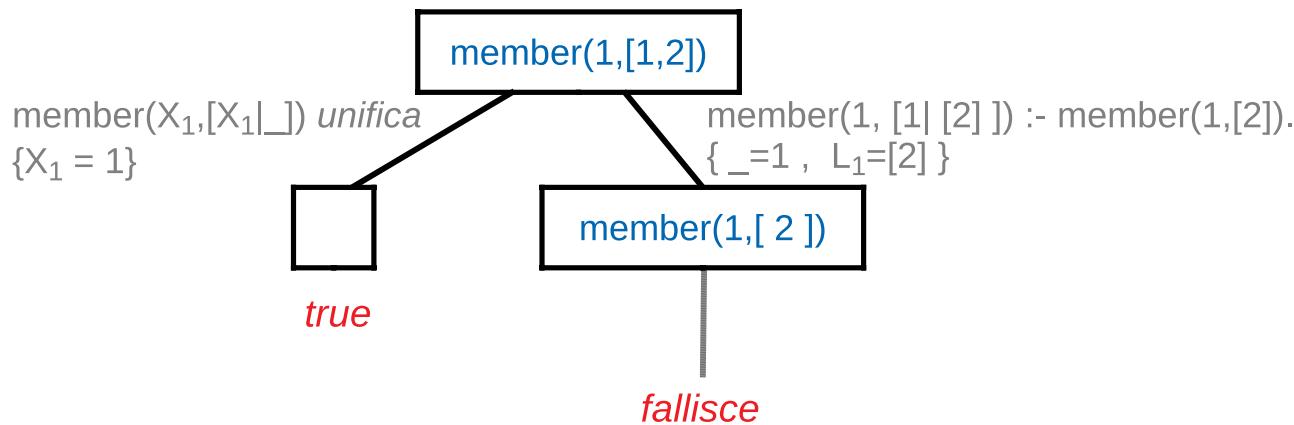
$\text{member}(X_1,[X_1|_])$ unifica
 $\{X_1 = 1\}$



```
member(X, [X | _]).  
member(X, [_ | L]) :- member(X, L).
```



Esempi di derivazione per member



Questo ramo non viene esplorato perché la query è ground

```
member(X, [X|_]).  
member(X, [_|L]) :- member(X,L).
```



Esempi di derivazione per member

member(2,[1,2])

```
member(X, [X | _]).  
member(X, [_ | L]) :- member(X, L).
```



Esempi di derivazione per member

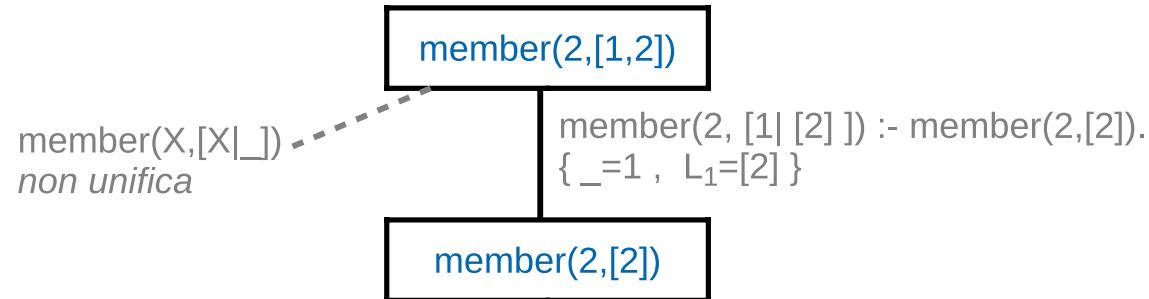
member(2,[1,2])

member(X,[X|_])
non unifica

```
member(X, [X | _]).  
member(X, [_ | L]) :- member(X, L).
```



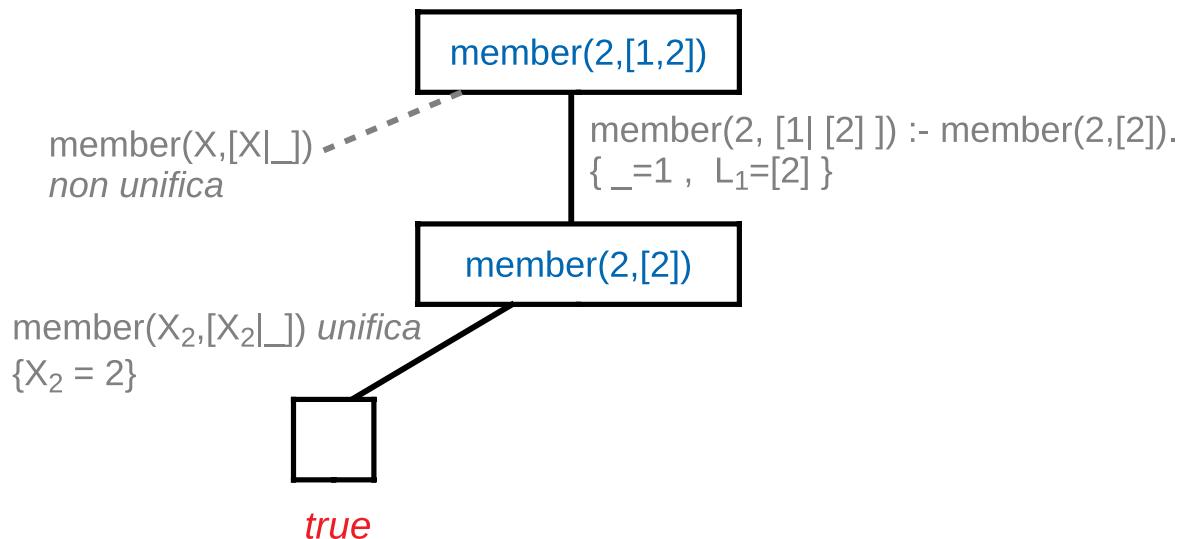
Esempi di derivazione per member



```
member(X, [X|_]).  
member(X, [_|L]) :- member(X,L).
```



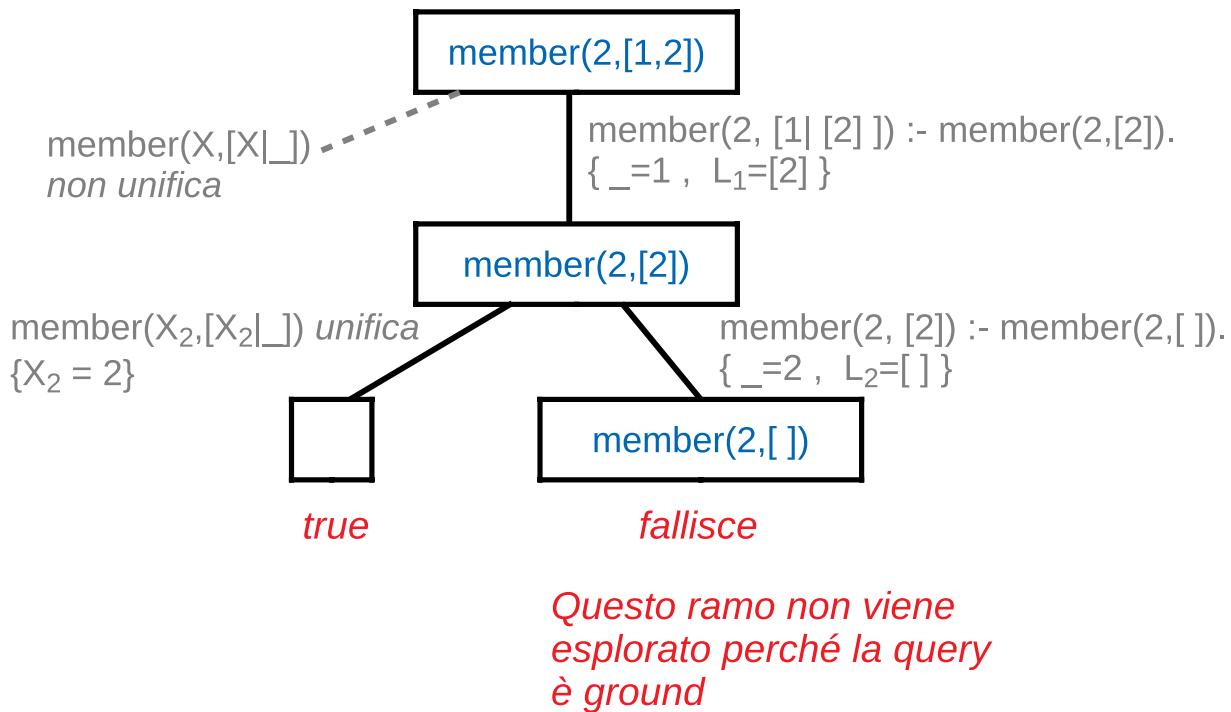
Esempi di derivazione per member



```
member(X, [X|_]).  
member(X, [_|L]) :- member(X, L).
```



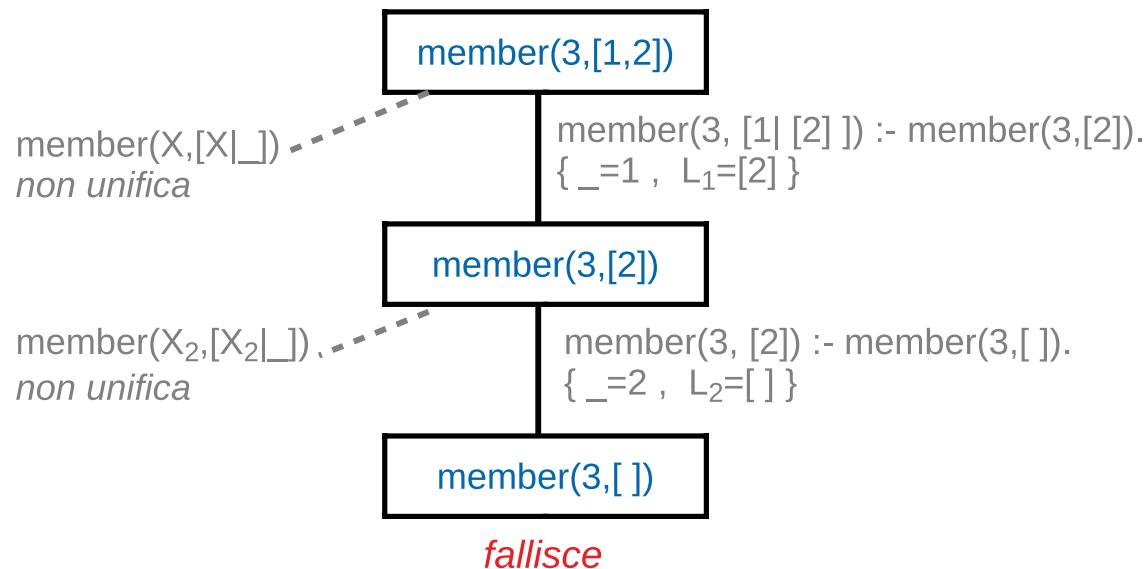
Esempi di derivazione per member



```
member(X, [X|_]).  
member(X, [_|L]) :- member(X,L).
```



Esempi di derivazione per member

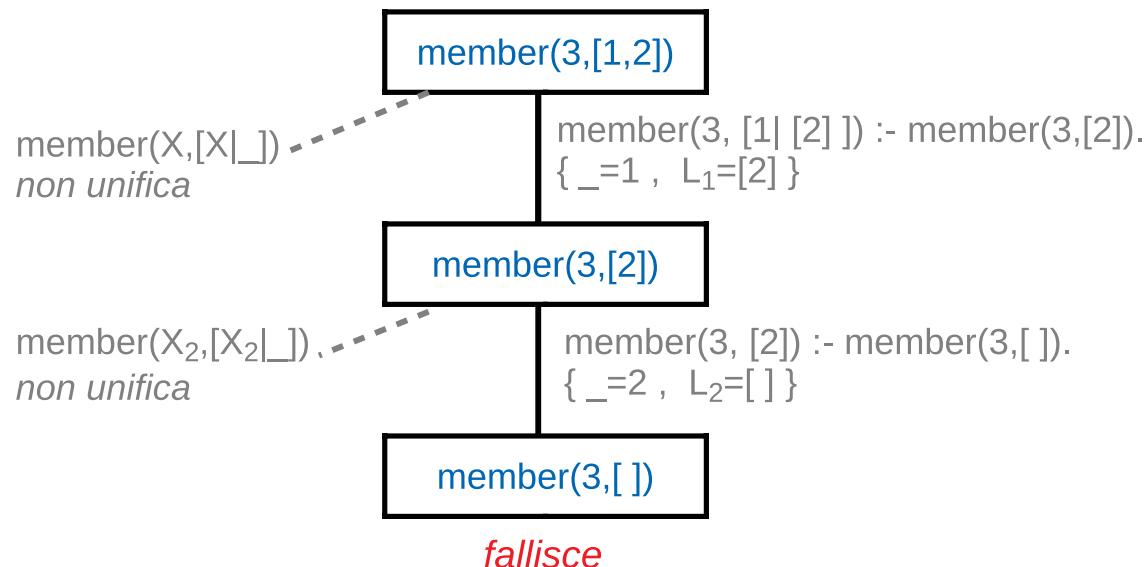


Non ci sono altri rami da provare
→ Risponde *false*

```
member(X,[X|_]).  
member(X,[_|L]) :- member(X,L).
```



Esempi di derivazione per member



Non ci sono altri rami da provare
→ Risponde *false*

```
member(X,[X|_]).  
member(X,[_|L]) :- member(X,L).
```

- Notare che in ML avrebbe sollevato una eccezione. Qui restituisce *false*



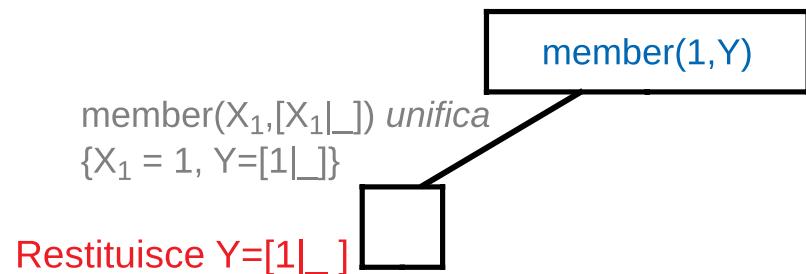
Esempi di derivazione per member

member(1,Y)

```
member(X,[X|_]).  
member(X,[_|L]) :- member(X,L).
```



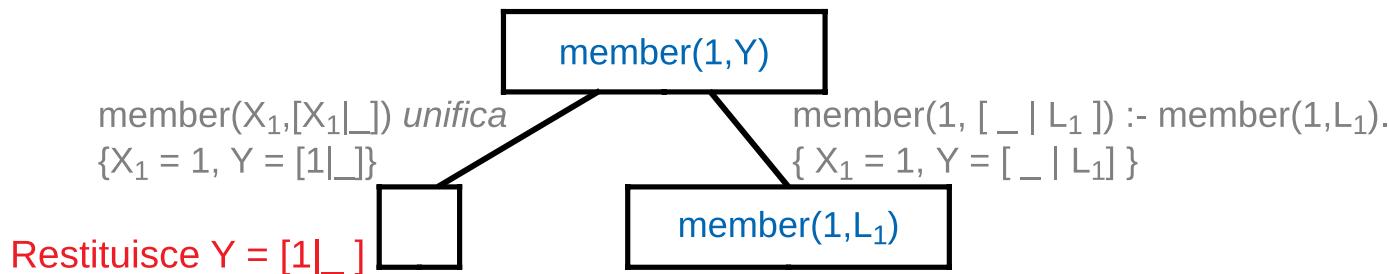
Esempi di derivazione per member



```
member(X,[X|_]).  
member(X,[_|L]) :- member(X,L).
```



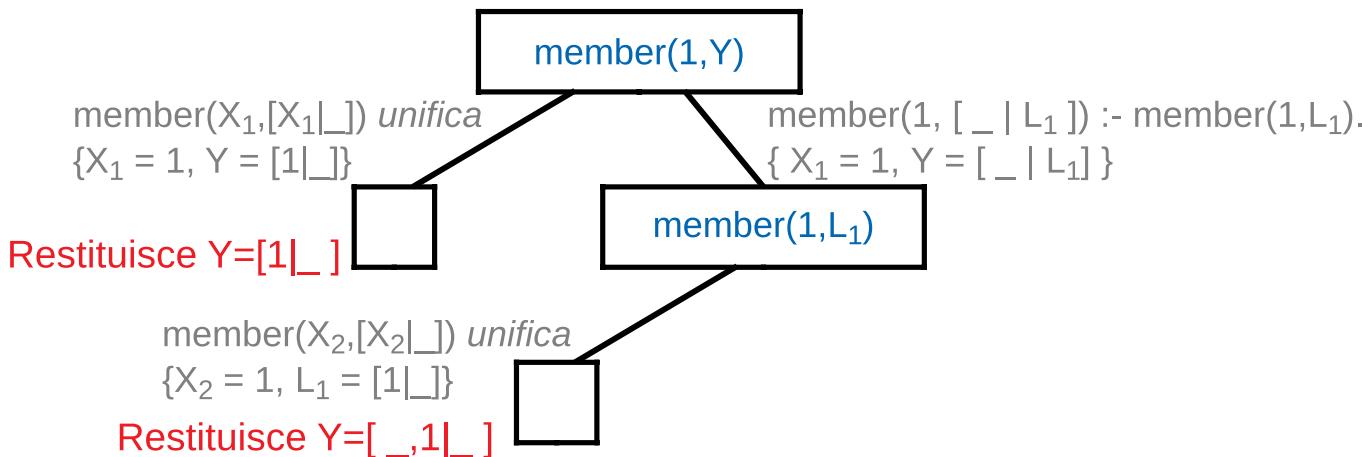
Esempi di derivazione per member



```
member(X,[X|_]).  
member(X,[_|L]) :- member(X,L).
```



Esempi di derivazione per member



```
member(X, [X | _]).  
member(X, [ _ | L]) :- member(X, L).
```

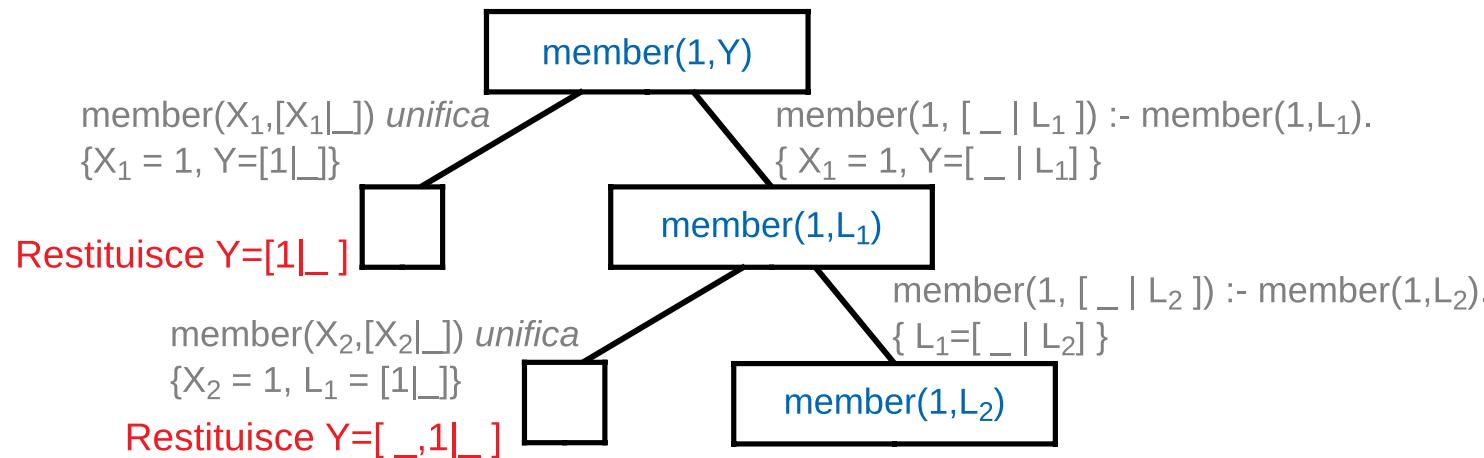
Il valore di Y si ottiene così:

$$L_1 = [1 | _],$$

$$Y = [_ | L_1] = [_ | [1 | _]] = [_, 1 | _]$$



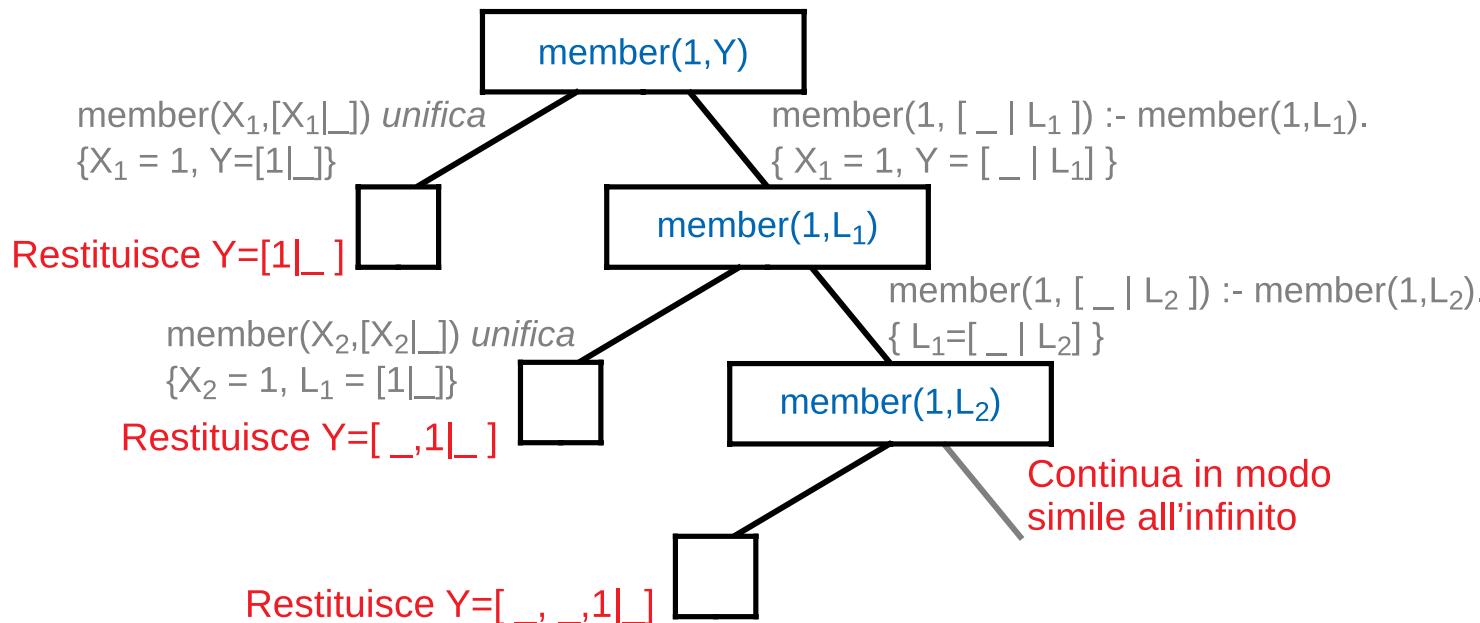
Esempi di derivazione per member



```
member(X, [X|_]).  
member(X, [_|L]) :- member(X, L).
```



Esempi di derivazione per member



```
member(X, [X | _]).  
member(X, [__ | L]) :- member(X, L).
```

Scrivete voi come si ottiene il valore di Y



Evanescenza di parametri di Input e Output

- Non c'è una chiara distinzione tra parametri IN e OUT
 - ◆ Ogni parametro può essere legato a un termine con costruttori (input)

[Paradigma logico](#)

[Prolog](#)

[Costrutti Prolog](#)

[Unificazione](#)

[Conjunctive queries](#)

[Le Regole](#)

[Ragionamento](#)

[Liste](#)

Sintassi

Il predicato member

Prolog vs ML

IN e OUT

Append

Negazione

[Applicazioni](#)

[Programmazione nondeterministica](#)

[Unicità di Prolog](#)



Evanescenza di parametri di Input e Output

■ Non c'è una chiara distinzione tra parametri IN e OUT

- ◆ Ogni parametro può essere legato a un termine con costruttori (input)
- ◆ Ogni parametro attuale con una variabile libera produce delle sostituzioni (output)

[Paradigma logico](#)

[Prolog](#)

[Costrutti Prolog](#)

[Unificazione](#)

[Conjunctive queries](#)

[Le Regole](#)

[Ragionamento](#)

[Liste](#)

Sintassi

Il predicato member

Prolog vs ML

IN e OUT

Append

Negazione

[Applicazioni](#)

Programmazione
nondeterministica

[Unicità di Prolog](#)



Evanescenza di parametri di Input e Output

- Non c'è una chiara distinzione tra parametri IN e OUT
 - ◆ Ogni parametro può essere legato a un termine con costruttori (input)
 - ◆ Ogni parametro attuale con una variabile libera produce delle sostituzioni (output)
 - ◆ I parametri con almeno un costruttore e una variabile forniscono sia un input sia un output

[Paradigma logico](#)

[Prolog](#)

[Costrutti Prolog](#)

[Unificazione](#)

[Conjunctive queries](#)

[Le Regole](#)

[Ragionamento](#)

[Liste](#)

Sintassi

Il predicato member

Prolog vs ML

IN e OUT

Append

Negazione

[Applicazioni](#)

Programmazione
nondeterministica

Unicità di Prolog



Evanescenza di parametri di Input e Output

[Paradigma logico](#)

[Prolog](#)

[Costrutti Prolog](#)

[Unificazione](#)

[Conjunctive queries](#)

[Le Regole](#)

[Ragionamento](#)

[Liste](#)

Sintassi

Il predicato member

Prolog vs ML

IN e OUT

Append

Negazione

[Applicazioni](#)

Programmazione
nondeterministica

Unicità di Prolog

- Non c'è una chiara distinzione tra parametri IN e OUT
 - ◆ Ogni parametro può essere legato a un termine con costruttori (input)
 - ◆ Ogni parametro attuale con una variabile libera produce delle sostituzioni (output)
 - ◆ I parametri con almeno un costruttore e una variabile forniscono sia un input sia un output
- Guardiamo gli esempi con member:
 - ◆ `member(X,<lista ground>)` prende una lista e restituisce i suoi elementi. Modalità: (OUT,IN)



Evanescenza di parametri di Input e Output

[Paradigma logico](#)

[Prolog](#)

[Costrutti Prolog](#)

[Unificazione](#)

[Conjunctive queries](#)

[Le Regole](#)

[Ragionamento](#)

[Liste](#)

Sintassi

Il predicato member

Prolog vs ML

IN e OUT

Append

Negazione

[Applicazioni](#)

Programmazione
nondeterministica

Unicità di Prolog

- Non c'è una chiara distinzione tra parametri IN e OUT
 - ◆ Ogni parametro può essere legato a un termine con costruttori (input)
 - ◆ Ogni parametro attuale con una variabile libera produce delle sostituzioni (output)
 - ◆ I parametri con almeno un costruttore e una variabile forniscono sia un input sia un output
- Guardiamo gli esempi con member:
 - ◆ `member(X,<lista ground>)` prende una lista e restituisce i suoi elementi. Modalità: (OUT,IN)
 - ◆ `member(<elem. ground>,L)` prende un elemento e restituisce le liste che lo conengono. Modalità: (IN,OUT)



Evanescenza di parametri di Input e Output

Paradigma logico

Prolog

Costrutti Prolog

Unificazione

Conjunctive queries

Le Regole

Ragionamento

Liste

Sintassi

Il predicato member

Prolog vs ML

IN e OUT

Append

Negazione

Applicazioni

Programmazione
nondeterministica

Unicità di Prolog

- Non c'è una chiara distinzione tra parametri IN e OUT
 - ◆ Ogni parametro può essere legato a un termine con costruttori (input)
 - ◆ Ogni parametro attuale con una variabile libera produce delle sostituzioni (output)
 - ◆ I parametri con almeno un costruttore e una variabile forniscono sia un input sia un output
- Guardiamo gli esempi con member:
 - ◆ `member(X,<lista ground>)` prende una lista e restituisce i suoi elementi. Modalità: (OUT,IN)
 - ◆ `member(<elem. ground>,L)` prende un elemento e restituisce le liste che lo conengono. Modalità: (IN,OUT)
 - ◆ *Invertibilità dei predicati*: possono rappresentare sia una funzione sia la sua inversa



Evanescenza di parametri di Input e Output

Paradigma logico

Prolog

Costrutti Prolog

Unificazione

Conjunctive queries

Le Regole

Ragionamento

Liste

Sintassi

Il predicato member

Prolog vs ML

IN e OUT

Append

Negazione

Applicazioni

Programmazione
nondeterministica

Unicità di Prolog

- Non c'è una chiara distinzione tra parametri IN e OUT
 - ◆ Ogni parametro può essere legato a un termine con costruttori (input)
 - ◆ Ogni parametro attuale con una variabile libera produce delle sostituzioni (output)
 - ◆ I parametri con almeno un costruttore e una variabile forniscono sia un input sia un output
- Guardiamo gli esempi con member:
 - ◆ `member(X,<lista ground>)` prende una lista e restituisce i suoi elementi. Modalità: (OUT,IN)
 - ◆ `member(<elem. ground>,L)` prende un elemento e restituisce le liste che lo conengono. Modalità: (IN,OUT)
 - ◆ *Invertibilità dei predici*: possono rappresentare sia una funzione sia la sua inversa
- Incontreremo lo stesso fenomeno in `append/3` (predicato append con 3 argomenti)



Esercizi

[Paradigma logico](#)

[Prolog](#)

[Costrutti Prolog](#)

[Unificazione](#)

[Conjunctive queries](#)

[Le Regole](#)

[Ragionamento](#)

[Liste](#)

Sintassi

Il predicato member

Prolog vs ML

IN e OUT

Append

Negazione

[Applicazioni](#)

Programmazione
nondeterministica

[Unicità di Prolog](#)

1. Data la lista [stud(mario,m), stud(maria,f), stud(paolo,m)]

(a) scrivere una opportuna query a member che restituisce gli studenti maschi



Esercizi

1. Data la lista [stud(mario,m), stud(maria,f), stud(paolo,m)]

- (a) scrivere una opportuna query a member che restituisce gli studenti maschi
- (b) disegnare il search tree per la query e la lista data.
- (c) scrivere le sostituzioni di risposta

Paradigma logico

Prolog

Costrutti Prolog

Unificazione

Conjunctive queries

Le Regole

Ragionamento

Liste

Sintassi

Il predicato member

Prolog vs ML

IN e OUT

Append

Negazione

Applicazioni

Programmazione
nondeterministica

Unicità di Prolog



Esercizi

Paradigma logico

Prolog

Costrutti Prolog

Unificazione

Conjunctive queries

Le Regole

Ragionamento

Liste

Sintassi

Il predicato member

Prolog vs ML

IN e OUT

Append

Negazione

Applicazioni

Programmazione
nondeterministica

Unicità di Prolog

1. Data la lista [stud(mario,m), stud(maria,f), stud(paolo,m)]
 - (a) scrivere una opportuna query a member che restituisce gli studenti maschi
 - (b) disegnare il search tree per la query e la lista data.
 - (c) scrivere le sostituzioni di risposta
2. Usando member, scrivere la regola per una select (algebra relazionale) sulla relazione r che seleziona i record il cui secondo elemento appartiene a una lista data. Se r ha n colonne, questa select ha n+1 argomenti:

```
select_r_2(X1, X2, ..., Xn, Lista)
```



Esercizi

Paradigma logico

Prolog

Costrutti Prolog

Unificazione

Conjunctive queries

Le Regole

Ragionamento

Liste

Sintassi

Il predicato member

Prolog vs ML

IN e OUT

Append

Negazione

Applicazioni

Programmazione nondeterministica

Unicità di Prolog

1. Data la lista [stud(mario,m), stud(maria,f), stud(paolo,m)]
 - (a) scrivere una opportuna query a member che restituisce gli studenti maschi
 - (b) disegnare il search tree per la query e la lista data.
 - (c) scrivere le sostituzioni di risposta
2. Usando member, scrivere la regola per una select (algebra relazionale) sulla relazione r che seleziona i record il cui secondo elemento appartiene a una lista data. Se r ha n colonne, questa select ha n+1 argomenti:

```
select_r_2(X1, X2, ..., Xn, Lista)
```

3. Scrivere un predicato reverse che inverte una lista. Prendere esempio dalla soluzione per ML che fa uso di un accumulatore per costruire progressivamente la lista invertita:

```
/* significato dei parametri del predicato */  
reverse(Lista, ListaInvertita, Accumulatore)
```

Poi usare l'overloading per avere una reverse a 2 soli argomenti.



Il predicato append

1. Scrivere un predicato append che concatena due liste:

```
/* schema */  
append( Lista1, Lista2, Risultato )
```

Ispirarsi alla soluzione ricorsiva per ML

[Paradigma logico](#)

[Prolog](#)

[Costrutti Prolog](#)

[Unificazione](#)

[Conjunctive queries](#)

[Le Regole](#)

[Ragionamento](#)

[Liste](#)

Sintassi

Il predicato member

Prolog vs ML

IN e OUT

Append

Negazione

[Applicazioni](#)

Programmazione
nondeterministica

Unicità di Prolog



Il predicato append

1. Scrivere un predicato append che concatena due liste:

```
/* schema */  
append( Lista1 , Lista2 , Risultato )
```

Ispirarsi alla soluzione ricorsiva per ML

2. Usare l'invertibilità di append per verificare se [a,b,c] è un prefisso di una lista data.

[Paradigma logico](#)

[Prolog](#)

[Costrutti Prolog](#)

[Unificazione](#)

[Conjunctive queries](#)

[Le Regole](#)

[Ragionamento](#)

[Liste](#)

Sintassi

Il predicato member

Prolog vs ML

IN e OUT

Append

Negazione

[Applicazioni](#)

Programmazione
nondeterministica

Unicità di Prolog



Il predicato append

1. Scrivere un predicato append che concatena due liste:

```
/* schema */  
append( Lista1 , Lista2 , Risultato )
```

Ispirarsi alla soluzione ricorsiva per ML

2. Usare l'invertibilità di append per verificare se [a,b,c] è un prefisso di una lista data.
 - verificare disegnando il search tree per la lista [a,b,c,d]

[Paradigma logico](#)

[Prolog](#)

[Costrutti Prolog](#)

[Unificazione](#)

[Conjunctive queries](#)

[Le Regole](#)

[Ragionamento](#)

[Liste](#)

Sintassi

Il predicato member

Prolog vs ML

IN e OUT

Append

Negazione

[Applicazioni](#)

Programmazione
nondeterministica

Unicità di Prolog



Il predicato append

1. Scrivere un predicato append che concatena due liste:

```
/* schema */  
append( Lista1 , Lista2 , Risultato )
```

Ispirarsi alla soluzione ricorsiva per ML

2. Usare l'invertibilità di append per verificare se [a,b,c] è un prefisso di una lista data.
 - verificare disegnando il search tree per la lista [a,b,c,d]
3. Usare l'invertibilità di append per verificare se [a,b,c] è un suffisso di una lista data.

[Paradigma logico](#)

[Prolog](#)

[Costrutti Prolog](#)

[Unificazione](#)

[Conjunctive queries](#)

[Le Regole](#)

[Ragionamento](#)

[Liste](#)

Sintassi

Il predicato member

Prolog vs ML

IN e OUT

Append

Negazione

[Applicazioni](#)

Programmazione
nondeterministica

Unicità di Prolog



Il predicato append

1. Scrivere un predicato append che concatena due liste:

```
/* schema */  
append( Lista1 , Lista2 , Risultato )
```

Ispirarsi alla soluzione ricorsiva per ML

2. Usare l'invertibilità di append per verificare se $[a,b,c]$ è un prefisso di una lista data.
 - verificare disegnando il search tree per la lista $[a,b,c,d]$
3. Usare l'invertibilità di append per verificare se $[a,b,c]$ è un suffisso di una lista data.
 - verificare disegnando il search tree per la lista $[a,a,b,c]$

Paradigma logico

Prolog

Costrutti Prolog

Unificazione

Conjunctive queries

Le Regole

Ragionamento

Liste

Sintassi

Il predicato member

Prolog vs ML

IN e OUT

Append

Negazione

Applicazioni

Programmazione
nondeterministica

Unicità di Prolog



Il predicato append

1. Scrivere un predicato append che concatena due liste:

```
/* schema */  
append( Lista1 , Lista2 , Risultato )
```

Ispirarsi alla soluzione ricorsiva per ML

2. Usare l'invertibilità di append per verificare se [a,b,c] è un prefisso di una lista data.
 - verificare disegnando il search tree per la lista [a,b,c,d]
3. Usare l'invertibilità di append per verificare se [a,b,c] è un suffisso di una lista data.
 - verificare disegnando il search tree per la lista [a,a,b,c]
4. Usare l'invertibilità di append per definire un predicato last(L,X) che è vero se X è l'ultimo elemento della lista L.

Paradigma logico

Prolog

Costrutti Prolog

Unificazione

Conjunctive queries

Le Regole

Ragionamento

Liste

Sintassi

Il predicato member

Prolog vs ML

IN e OUT

Append

Negazione

Applicazioni

Programmazione nondeterministica

Unicità di Prolog



Ancora append

■ Generazione di tutti i prefissi / suffissi di una lista

```
prefix(Prefix, List) :- append(Prefix, _, List).
```

```
suffix(Suffix, List) :- append(_, Suffix, List).
```

[Paradigma logico](#)

[Prolog](#)

[Costrutti Prolog](#)

[Unificazione](#)

[Conjunctive queries](#)

[Le Regole](#)

[Ragionamento](#)

[Liste](#)

Sintassi

Il predicato member

Prolog vs ML

IN e OUT

Append

Negazione

[Applicazioni](#)

Programmazione
nondeterministica

[Unicità di Prolog](#)



Ancora append

Paradigma logico

Prolog

Costrutti Prolog

Unificazione

Conjunctive queries

Le Regole

Ragionamento

Liste

Sintassi

Il predicato member

Prolog vs ML

IN e OUT

Append

Negazione

Applicazioni

Programmazione
nondeterministica

Unicità di Prolog

■ Generazione di tutti i prefissi / suffissi di una lista

```
prefix(Prefix, List) :- append(Prefix, _, List).
```

```
suffix(Suffix, List) :- append(_, Suffix, List).
```

■ Definizione alternativa di member attraverso append

```
member(X, L) :- append(_, [X|_], L).
```



Ancora append

Paradigma logico

Prolog

Costrutti Prolog

Unificazione

Conjunctive queries

Le Regole

Ragionamento

Liste

Sintassi

Il predicato member

Prolog vs ML

IN e OUT

Append

Negazione

Applicazioni

Programmazione nondeterministica

Unicità di Prolog

■ Generazione di tutti i prefissi / suffissi di una lista

```
prefix(Prefix, List) :- append(Prefix, _, List).  
  
suffix(Suffix, List) :- append(_, Suffix, List).
```

■ Definizione alternativa di member attraverso append

```
member(X, L) :- append(_, [X|_], L).
```

■ Calcolo delle sottoliste di L. Notare che S è una sottolista di L sse valgono queste due condizioni (equivalenti)

- ◆ S è il prefisso di un suffisso di L
- ◆ S è il suffisso di un prefisso di L

```
sublist(Sub, L) :- prefix(Pre, List), suffix(Sub, Pre).
```



Ancora cammini su grafi

- Ora possiamo definire i cammini su un grafo, supportando anche cicli
- Definiamo il predicato `connessi_evitando(X,Y,Avoid)`: c'è un cammino da X a Y che evita i nodi nella lista Avoid

```
connessi_evitando(X,X,Avoid) :- \+ member(X,Avoid).  
connessi_evitando(X,Y,Avoid) :- arco(X,Z),  
                                \+ member(Z,Avoid),  
                                connessi_evitando(Z,Y,[X|Avoid])
```

[Paradigma logico](#)

[Prolog](#)

[Costrutti Prolog](#)

[Unificazione](#)

[Conjunctive queries](#)

[Le Regole](#)

[Ragionamento](#)

[Liste](#)

[Sintassi](#)

[Il predicato member](#)

[Prolog vs ML](#)

[IN e OUT](#)

[Append](#)

[Negazione](#)

[Applicazioni](#)

[Programmazione nondeterministica](#)

[Unicità di Prolog](#)



Ancora cammini su grafi

[Paradigma logico](#)

[Prolog](#)

[Costrutti Prolog](#)

[Unificazione](#)

[Conjunctive queries](#)

[Le Regole](#)

[Ragionamento](#)

[Liste](#)

[Sintassi](#)

[Il predicato member](#)

[Prolog vs ML](#)

[IN e OUT](#)

[Append](#)

[Negazione](#)

[Applicazioni](#)

[Programmazione nondeterministica](#)

[Unicità di Prolog](#)

- Ora possiamo definire i cammini su un grafo, supportando anche cicli
- Definiamo il predicato `connessi_evitando(X,Y,Avoid)`: c'è un cammino da X a Y che evita i nodi nella lista Avoid

```
connessi_evitando(X,X,Avoid) :- \+ member(X,Avoid).  
connessi_evitando(X,Y,Avoid) :- arco(X,Z),  
                                \+ member(Z,Avoid),  
                                connessi_evitando(Z,Y,[X|Avoid])
```

- Ora (ri)definiamo la connessione tra due nodi:

```
connessi3(X,Y) :- connessi_evitando(X,Y,[]).
```

`connessi3` termina sempre, anche su grafi dotati di cicli



[Paradigma logico](#)

[Prolog](#)

[Costrutti Prolog](#)

[Unificazione](#)

[Conjunctive queries](#)

[Le Regole](#)

[Ragionamento](#)

[Liste](#)

[**Applicazioni**](#)

Aritmetica in Prolog

Derivate

Derivate

Uguaglianza

Aritmetica

Programmazione

nondeterministica

Unicità di Prolog

Calcolo simbolico in Prolog



Aritmetica in Prolog

[Paradigma logico](#)

[Prolog](#)

[Costrutti Prolog](#)

[Unificazione](#)

[Conjunctive queries](#)

[Le Regole](#)

[Ragionamento](#)

[Liste](#)

[Applicazioni](#)

[Aritmetica in Prolog](#)

Derivate

Derivate

Uguaglianza

Aritmetica

Programmazione
nondeterministica

Unicità di Prolog

■ Operatori aritmetici:

+ , - , *

/, divisione con risultato float

//, divisione con risultato intero

mod

**, elevamento a potenza (anche tra float)

■ Operatori relazionali:

< , <= , > , >=



Aritmetica

[Paradigma logico](#)

[Prolog](#)

[Costrutti Prolog](#)

[Unificazione](#)

[Conjunctive queries](#)

[Le Regole](#)

[Ragionamento](#)

[Liste](#)

[Applicazioni](#)

[Aritmetica in Prolog](#)

[Derivate](#)

[Derivate](#)

[Uguaglianza](#)

[Aritmetica](#)

[Programmazione
nondeterministica](#)

[Unicità di Prolog](#)

- In Prolog gli operatori aritmetici sono *costruttori* (aka *funtori*)
 - ◆ $3+5*2$ *non* è uguale a 13!



Aritmetica

[Paradigma logico](#)

[Prolog](#)

[Costrutti Prolog](#)

[Unificazione](#)

[Conjunctive queries](#)

[Le Regole](#)

[Ragionamento](#)

[Liste](#)

[Applicazioni](#)

[Aritmetica in Prolog](#)

[Derivate](#)

Derivate

Uguaglianza

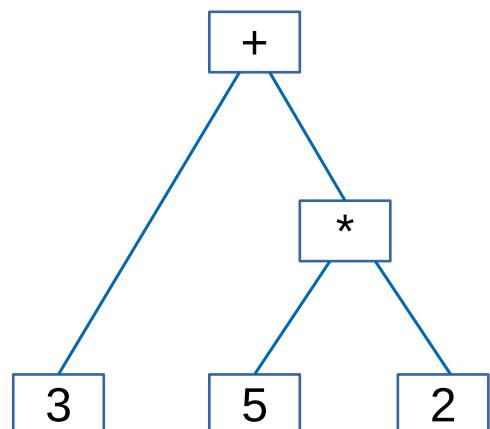
Aritmetica

[Programmazione
nondeterministica](#)

[Unicità di Prolog](#)

■ In Prolog gli operatori aritmetici sono *costruttori* (aka *funtori*)

- ◆ $3+5*2$ *non* è uguale a 13!
- ◆ è un termine con albero sintattico qui a destra
- ◆ Per calcolare $3+5*2$ usare l'operatore **is**:
Ris is 3+5*2
- ◆ questo goal è vero se Ris è uguale al *valore* di $3+5*2$





Aritmetica

[Paradigma logico](#)

[Prolog](#)

[Costrutti Prolog](#)

[Unificazione](#)

[Conjunctive queries](#)

[Le Regole](#)

[Ragionamento](#)

[Liste](#)

[Applicazioni](#)

[Aritmetica in Prolog](#)

[Derivate](#)

Derivate

Uguaglianza

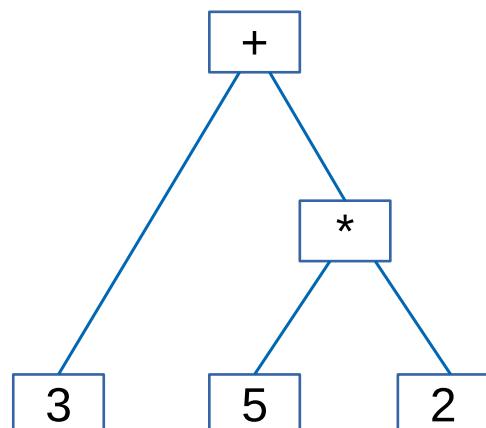
Aritmetica

[Programmazione
nondeterministica](#)

[Unicità di Prolog](#)

- In Prolog gli operatori aritmetici sono *costruttori* (aka *funtori*)

- ◆ $3+5*2$ *non* è uguale a 13!
 - ◆ è un termine con albero sintattico qui a destra
 - ◆ Per calcolare $3+5*2$ usare l'operatore **is**:
Ris is 3+5*2
 - ◆ questo goal è vero se Ris è uguale al *valore* di $3+5*2$



- Questo rende il calcolo simbolico particolarmente semplice. Facciamo un esempio basato sulle derivate, in stile Mathematica



Calcolo simbolico delle derivate

[Paradigma logico](#)

[Prolog](#)

[Costrutti Prolog](#)

[Unificazione](#)

[Conjunctive queries](#)

[Le Regole](#)

[Ragionamento](#)

[Liste](#)

[Applicazioni](#)

[Aritmetica in Prolog](#)

Derivate

Derivate

Uguaglianza

Aritmetica

[Programmazione
nondeterministica](#)

[Unicità di Prolog](#)

Schema del predicato:

der(Expr,X,D) è vero se D è la derivata di Expr rispetto a X

der(X, X, 1).



Calcolo simbolico delle derivate

[Paradigma logico](#)

[Prolog](#)

[Costrutti Prolog](#)

[Unificazione](#)

[Conjunctive queries](#)

[Le Regole](#)

[Ragionamento](#)

[Liste](#)

[Applicazioni](#)

[Aritmetica in Prolog](#)

[Derivate](#)

Derivate

Uguaglianza

Aritmetica

[Programmazione
nondeterministica](#)

[Unicità di Prolog](#)

Schema del predicato:

der(Expr,X,D) è vero se D è la derivata di Expr rispetto a X

```
der(X, X, 1).  
der(X^N, X, N*X^N1) :-  
    N1 is N-1.
```



Calcolo simbolico delle derivate

[Paradigma logico](#)

[Prolog](#)

[Costrutti Prolog](#)

[Unificazione](#)

[Conjunctive queries](#)

[Le Regole](#)

[Ragionamento](#)

[Liste](#)

[Applicazioni](#)

[Aritmetica in Prolog](#)

[Derivate](#)

Derivate

Uguaglianza

Aritmetica

[Programmazione
nondeterministica](#)

[Unicità di Prolog](#)

Schema del predicato:

der(Expr,X,D) è vero se D è la derivata di Expr rispetto a X

```
der(X, X, 1).  
der(X^N, X, N*X^N1) :-  
          N1 is N-1.  
der(sen(X), X, cos(X)).
```



Calcolo simbolico delle derivate

[Paradigma logico](#)

[Prolog](#)

[Costrutti Prolog](#)

[Unificazione](#)

[Conjunctive queries](#)

[Le Regole](#)

[Ragionamento](#)

[Liste](#)

[Applicazioni](#)

[Aritmetica in Prolog](#)

[Derivate](#)

Derivate

Uguaglianza

Aritmetica

[Programmazione
nondeterministica](#)

[Unicità di Prolog](#)

Schema del predicato:

der(Expr,X,D) è vero se D è la derivata di Expr rispetto a X

```
der(X, X, 1).  
der(X^N, X, N*X^N1) :-  
    N1 is N-1.  
der(sen(X), X, cos(X)).  
der(cos(X), X, -sen(X)).
```



Calcolo simbolico delle derivate

[Paradigma logico](#)

[Prolog](#)

[Costrutti Prolog](#)

[Unificazione](#)

[Conjunctive queries](#)

[Le Regole](#)

[Ragionamento](#)

[Liste](#)

[Applicazioni](#)

[Aritmetica in Prolog](#)

Derivate

Derivate

Uguaglianza

Aritmetica

[Programmazione
nondeterministica](#)

[Unicità di Prolog](#)

Schema del predicato:

der(Expr,X,D) è vero se D è la derivata di Expr rispetto a X

```
der(X, X, 1).  
der(X^N, X, N*X^N1) :-  
    N1 is N-1.  
der(sen(X), X, cos(X)).  
der(cos(X), X, -sen(X)).  
der(log(X), X, 1/X).
```



Calcolo simbolico delle derivate

[Paradigma logico](#)

[Prolog](#)

[Costrutti Prolog](#)

[Unificazione](#)

[Conjunctive queries](#)

[Le Regole](#)

[Ragionamento](#)

[Liste](#)

[Applicazioni](#)

[Aritmetica in Prolog](#)

Derivate

Derivate

Uguaglianza

Aritmetica

Programmazione
nondeterministica

Unicità di Prolog

Schema del predicato:

der(Expr,X,D) è vero se D è la derivata di Expr rispetto a X

```
der(X, X, 1).  
der(X^N, X, N*X^N1) :-  
    N1 is N-1.  
der(sen(X), X, cos(X)).  
der(cos(X), X, -sen(X)).  
der(log(X), X, 1/X).  
der(F+G, X, DF+DG) :-  
    der(F, X, DF), der(G, X, DG).
```



Calcolo simbolico delle derivate

[Paradigma logico](#)

[Prolog](#)

[Costrutti Prolog](#)

[Unificazione](#)

[Conjunctive queries](#)

[Le Regole](#)

[Ragionamento](#)

[Liste](#)

[Applicazioni](#)

[Aritmetica in Prolog](#)

[Derivate](#)

[Derivate](#)

[Uguaglianza](#)

[Aritmetica](#)

[Programmazione
nondeterministica](#)

[Unicità di Prolog](#)

Schema del predicato:

der(Expr,X,D) è vero se D è la derivata di Expr rispetto a X

```
der(X, X, 1).  
der(X^N, X, N*X^N1) :-  
    N1 is N-1.  
der(sen(X), X, cos(X)).  
der(cos(X), X, -sen(X)).  
der(log(X), X, 1/X).  
der(F+G, X, DF+DG) :-  
    der(F, X, DF), der(G, X, DG).  
der(F-G, X, DF-DG) :-  
    der(F, X, DF), der(G, X, DG).
```



Calcolo simbolico delle derivate

[Paradigma logico](#)

[Prolog](#)

[Costrutti Prolog](#)

[Unificazione](#)

[Conjunctive queries](#)

[Le Regole](#)

[Ragionamento](#)

[Liste](#)

[Applicazioni](#)

[Aritmetica in Prolog](#)

[Derivate](#)

[Derivate](#)

[Uguaglianza](#)

[Aritmetica](#)

[Programmazione
nondeterministica](#)

[Unicità di Prolog](#)

Schema del predicato:

der(Expr,X,D) è vero se D è la derivata di Expr rispetto a X

```
der(X, X, 1).  
der(X^N, X, N*X^N1) :-  
    N1 is N-1.  
der(sen(X), X, cos(X)).  
der(cos(X), X, -sen(X)).  
der(log(X), X, 1/X).  
der(F+G, X, DF+DG) :-  
    der(F, X, DF), der(G, X, DG).  
der(F-G, X, DF-DG) :-  
    der(F, X, DF), der(G, X, DG).  
der(F*G, X, F*Dg+Df*G) :-  
    der(F, X, DF), der(G, X, DG).  
...
```



Calcolo simbolico delle derivate

[Paradigma logico](#)

[Prolog](#)

[Costrutti Prolog](#)

[Unificazione](#)

[Conjunctive queries](#)

[Le Regole](#)

[Ragionamento](#)

[Liste](#)

[Applicazioni](#)

[Aritmetica in Prolog](#)

[Derivate](#)

Derivate

Uguaglianza

Aritmetica

[Programmazione
nondeterministica](#)

[Unicità di Prolog](#)

Schema del predicato:

der(Expr,X,D) è vero se D è la derivata di Expr rispetto a X

```
der(X, X, 1).  
der(X^N, X, N*X^N1) :-  
    N1 is N-1.  
der(sen(X), X, cos(X)).  
der(cos(X), X, -sen(X)).  
der(log(X), X, 1/X).  
der(F+G, X, DF+DG) :-  
    der(F, X, DF), der(G, X, DG).  
der(F-G, X, DF-DG) :-  
    der(F, X, DF), der(G, X, DG).  
der(F*G, X, F*DG+DF*G) :-  
    der(F, X, DF), der(G, X, DG).  
...
```

```
?- der(x^3*cos(x), x, D).
```

```
D = x^3* -sen(x)+3*x^2*cos(x).
```

[Si possono aggiungere predicati per semplificare il risultato]



Operatori di Uguaglianza tra Termini

Paradigma logico

Prolog

Costrutti Prolog

Unificazione

Conjunctive queries

Le Regole

Ragionamento

Liste

Applicazioni

Aritmetica in Prolog

Derivate

Derivate

Uguaglianza

Aritmetica

Programmazione
nondeterministica

Unicità di Prolog

Sintassi	Significato	Negazione
Term = Term	Sono unificabili	\=
Term == Term	Sono identici	\==

Esempi:

```
?- A = ciao.  
A = ciao.
```

```
?- A == ciao.  
false.
```

```
?- 4*3 = 3*4.  
false.
```

```
?- A = 3*4.  
A = 3*4.
```



Uguaglianza Aritmetica

[Paradigma logico](#)

[Prolog](#)

[Costrutti Prolog](#)

[Unificazione](#)

[Conjunctive queries](#)

[Le Regole](#)

[Ragionamento](#)

[Liste](#)

[Applicazioni](#)

[Aritmetica in Prolog](#)

Derivate

Derivate

[Uguaglianza](#)

Aritmetica

[Programmazione nondeterministica](#)

[Unicità di Prolog](#)

Qui, Expr deve essere un termine aritmetico *completamente istanziato*

Sintassi	Significato	Negazione
Term is Expr	Ha lo stesso valore numerico di (valuta l'espressione, poi unifica con il termine)	

Esempi:

```
?- 4*3 is 10+2.  
false.
```

```
?- 4*3 = 10+2.  
false.
```

```
? A is 10+2.  
A = 12.
```

```
? 12 is 10+B.  
ERROR: Arguments are not sufficiently instantiated
```

```
? A = 3, B is A+1.  
A=3,  
B=4.
```



Esercizi

[Paradigma logico](#)

[Prolog](#)

[Costrutti Prolog](#)

[Unificazione](#)

[Conjunctive queries](#)

[Le Regole](#)

[Ragionamento](#)

[Liste](#)

[Applicazioni](#)

Aritmetica in Prolog

Derivate

Derivate

Uguaglianza

Aritmetica

[Programmazione
nondeterministica](#)

[Unicità di Prolog](#)

Usando `is` definire

1. un predicato `len(L)` che conta gli elementi della lista `L`



Esercizi

[Paradigma logico](#)

[Prolog](#)

[Costrutti Prolog](#)

[Unificazione](#)

[Conjunctive queries](#)

[Le Regole](#)

[Ragionamento](#)

[Liste](#)

[Applicazioni](#)

Aritmetica in Prolog

Derivate

Derivate

Uguaglianza

Aritmetica

[Programmazione
nondeterministica](#)

[Unicità di Prolog](#)

Usando `is` definire

1. un predicato `len(L)` che conta gli elementi della lista `L`
2. un predicato `fatt(N,F)` tale che `F` è il fattoriale di `N`



Esercizi

[Paradigma logico](#)

[Prolog](#)

[Costrutti Prolog](#)

[Unificazione](#)

[Conjunctive queries](#)

[Le Regole](#)

[Ragionamento](#)

[Liste](#)

[Applicazioni](#)

Aritmetica in Prolog

Derivate

Derivate

Uguaglianza

Aritmetica

[Programmazione
nondeterministica](#)

[Unicità di Prolog](#)

Usando `is` definire

1. un predicato `len(L)` che conta gli elementi della lista `L`
2. un predicato `fatt(N,F)` tale che `F` è il fattoriale di `N`
3. un predicato `sum(L,S)` tale che `S` è la somma degli elementi nella lista `L`



Programmazione nondeterministica

[Paradigma logico](#)

[Prolog](#)

[Costrutti Prolog](#)

[Unificazione](#)

[Conjunctive queries](#)

[Le Regole](#)

[Ragionamento](#)

[Liste](#)

[Applicazioni](#)

[Programmazione
nondeterministica](#)

[In che consiste](#)

Tic tac toe

Analisi del gioco

I/O elementare

GUI

[Unicità di Prolog](#)

■ In cosa consiste:

- ◆ invece di dire a Prolog *come* trovare la soluzione di un problema
- ◆ si dice *cosa* è una soluzione e si lascia che Prolog la cerchi con il backtrack (visita del search tree)



Programmazione nondeterministica

Paradigma logico

Prolog

Costrutti Prolog

Unificazione

Conjunctive queries

Le Regole

Ragionamento

Liste

Applicazioni

Programmazione
nondeterministica

In che consiste

Tic tac toe

Analisi del gioco

I/O elementare

GUI

Unicità di Prolog

■ In cosa consiste:

- ◆ invece di dire a Prolog *come* trovare la soluzione di un problema
- ◆ si dice *cosa* è una soluzione e si lascia che Prolog la cerchi con il backtrack (visita del search tree)

■ Schema generale (*generate and test*):

```
solution(X) :- generate(X), test(X).
```



Programmazione nondeterministica

[Paradigma logico](#)

[Prolog](#)

[Costrutti Prolog](#)

[Unificazione](#)

[Conjunctive queries](#)

[Le Regole](#)

[Ragionamento](#)

[Liste](#)

[Applicazioni](#)

[Programmazione
nondeterministica](#)

[In che consiste](#)

Tic tac toe

Analisi del gioco

I/O elementare

GUI

[Unicità di Prolog](#)

■ In cosa consiste:

- ◆ invece di dire a Prolog *come* trovare la soluzione di un problema
- ◆ si dice *cosa* è una soluzione e si lascia che Prolog la cerchi con il backtrack (visita del search tree)

■ Schema generale (*generate and test*):

```
solution(X) :- generate(X), test(X).
```

- ◆ Prolog genera via via i candidati a soluzione X



Programmazione nondeterministica

[Paradigma logico](#)

[Prolog](#)

[Costrutti Prolog](#)

[Unificazione](#)

[Conjunctive queries](#)

[Le Regole](#)

[Ragionamento](#)

[Liste](#)

[Applicazioni](#)

[Programmazione
nondeterministica](#)

[In che consiste](#)

Tic tac toe

Analisi del gioco

I/O elementare

GUI

[Unicità di Prolog](#)

■ In cosa consiste:

- ◆ invece di dire a Prolog *come* trovare la soluzione di un problema
- ◆ si dice *cosa* è una soluzione e si lascia che Prolog la cerchi con il backtrack (visita del search tree)

■ Schema generale (*generate and test*):

```
solution(X) :- generate(X), test(X).
```

- ◆ Prolog genera via via i candidati a soluzione X
- ◆ per ciascuno di essi verifica se è effettivamente una soluzione (test)



Programmazione nondeterministica

Paradigma logico

Prolog

Costrutti Prolog

Unificazione

Conjunctive queries

Le Regole

Ragionamento

Liste

Applicazioni

Programmazione
nondeterministica

In che consiste

Tic tac toe

Analisi del gioco

I/O elementare

GUI

Unicità di Prolog

■ In cosa consiste:

- ◆ invece di dire a Prolog *come* trovare la soluzione di un problema
- ◆ si dice *cosa* è una soluzione e si lascia che Prolog la cerchi con il backtrack (visita del search tree)

■ Schema generale (*generate and test*):

```
solution(X) :- generate(X), test(X).
```

- ◆ Prolog genera via via i candidati a soluzione X
- ◆ per ciascuno di essi verifica se è effettivamente una soluzione (test)
- ◆ se sì, restituisce la soluzione; altrimenti fa backtrack e torna a generare il successivo candidato



Programmazione nondeterministica

Paradigma logico

Prolog

Costrutti Prolog

Unificazione

Conjunctive queries

Le Regole

Ragionamento

Liste

Applicazioni

Programmazione
nondeterministica

In che consiste

Tic tac toe

Analisi del gioco

I/O elementare

GUI

Unicità di Prolog

■ In cosa consiste:

- ◆ invece di dire a Prolog *come* trovare la soluzione di un problema
- ◆ si dice *cosa* è una soluzione e si lascia che Prolog la cerchi con il backtrack (visita del search tree)

■ Schema generale (*generate and test*):

```
solution(X) :- generate(X), test(X).
```

- ◆ Prolog genera via via i candidati a soluzione X
- ◆ per ciascuno di essi verifica se è effettivamente una soluzione (test)
- ◆ se sì, restituisce la soluzione; altrimenti fa backtrack e torna a generare il successivo candidato
- ◆ se si chiedono altre risposte, genera altre soluzioni



Confronto con gli altri paradigmi

Paradigma logico

Prolog

Costrutti Prolog

Unificazione

Conjunctive queries

Le Regole

Ragionamento

Liste

Applicazioni

Programmazione
nondeterministica

In che consiste

Tic tac toe

Analisi del gioco

I/O elementare

GUI

Unicità di Prolog

- La programmazione nondeterministica è una caratteristica unica del paradigma logico
- Come verrebbe approssimata negli altri paradigmi:

```
/* paradigma imperativo */
X := primo candidato;
while not test(X) and X ≠ null do
    X := prossimo_candidato(X)
return X
```



Confronto con gli altri paradigmi

Paradigma logico

Prolog

Costrutti Prolog

Unificazione

Conjunctive queries

Le Regole

Ragionamento

Liste

Applicazioni

Programmazione
nondeterministica

In che consiste

Tic tac toe

Analisi del gioco

I/O elementare

GUI

Unicità di Prolog

- La programmazione nondeterministica è una caratteristica unica del paradigma logico
- Come verrebbe approssimata negli altri paradigmi:

```
/* paradigma imperativo */
X := primo candidato;
while not test(X) and X ≠ null do
    X := prossimo_candidato(X)
return X

/* paradigma funzionale */
fun soluzione( X ) =
    if test(X) then X
    else soluzione( prossimo_candidato(X) )
```



Confronto con gli altri paradigmi

Paradigma logico

Prolog

Costrutti Prolog

Unificazione

Conjunctive queries

Le Regole

Ragionamento

Liste

Applicazioni

Programmazione
nondeterministica

In che consiste

Tic tac toe

Analisi del gioco

I/O elementare

GUI

Unicità di Prolog

- La programmazione nondeterministica è una caratteristica unica del paradigma logico
- Come verrebbe approssimata negli altri paradigmi:

```
/* paradigma imperativo */
X := primo_candidato;
while not test(X) and X ≠ null do
    X := prossimo_candidato(X)
return X

/* paradigma funzionale */
fun soluzione( X ) =
    if test(X) then X
    else soluzione( prossimo_candidato(X) )

/* invocare così */
soluzione( primo_candidato );
```



Confronto con gli altri paradigmi

Paradigma logico

Prolog

Costrutti Prolog

Unificazione

Conjunctive queries

Le Regole

Ragionamento

Liste

Applicazioni

Programmazione
nondeterministica

In che consiste

Tic tac toe

Analisi del gioco

I/O elementare

GUI

Unicità di Prolog

- La programmazione nondeterministica è una caratteristica unica del paradigma logico
- Come verrebbe approssimata negli altri paradigmi:

```
/* paradigma imperativo */
X := primo_candidato;
while not test(X) and X ≠ null do
    X := prossimo_candidato(X)
return X

/* paradigma funzionale */
fun soluzione( X ) =
    if test(X) then X
    else soluzione( prossimo_candidato(X) )

/* invocare così */
soluzione( primo_candidato );
```

- In entrambi i casi verrebbe generata solo la prima soluzione trovata (se si vogliono le altre occorre complicare il codice)
 - ◆ l'utilità del generare tutte le soluzioni sarà illustrata programmando l'AI di un gioco



Esempi di programmazione nondeterministica

[Paradigma logico](#)

[Prolog](#)

[Costrutti Prolog](#)

[Unificazione](#)

[Conjunctive queries](#)

[Le Regole](#)

[Ragionamento](#)

[Liste](#)

[Applicazioni](#)

[Programmazione
nondeterministica](#)

[In che consiste](#)

Tic tac toe

Analisi del gioco

I/O elementare

GUI

[Unicità di Prolog](#)

Ricerca dei membri pari di una lista

```
/* stile generate and test */
membro_pari(X,L) :- member(X,L), 0 is X mod 2.
```



Esempi di programmazione nondeterministica

Paradigma logico

Prolog

Costrutti Prolog

Unificazione

Conjunctive queries

Le Regole

Ragionamento

Liste

Applicazioni

Programmazione
nondeterministica

In che consiste

Tic tac toe

Analisi del gioco

I/O elementare

GUI

Unicità di Prolog

■ Ricerca dei membri pari di una lista

```
/* stile generate and test */
membro_pari(X,L) :- member(X,L), 0 is X mod 2.

/* invece di ricorsione ad hoc */
membro_pari(X,[X|_]) :- 0 is X mod 2.
membro_pari(X,[_|Resto]) :- membro_pari(X,Resto).
```



Esempi di programmazione nondeterministica

Paradigma logico

Prolog

Costrutti Prolog

Unificazione

Conjunctive queries

Le Regole

Ragionamento

Liste

Applicazioni

Programmazione
nondeterministica

In che consiste

Tic tac toe

Analisi del gioco

I/O elementare

GUI

Unicità di Prolog

■ Ricerca dei membri pari di una lista

```
/* stile generate and test */
membro_pari(X,L) :- member(X,L), 0 is X mod 2.

/* invece di ricorsione ad hoc */
membro_pari(X,[X|_]) :- 0 is X mod 2.
membro_pari(X,[_|Resto]) :- membro_pari(X,Resto).
```

- Notare come l'approccio generate & test possa giocare il ruolo delle funzioni di ordine superiore in ML
 - ◆ permette di comporre facilmente nuovi predicati da quelli dati



Esempi di programmazione nondeterministica

Paradigma logico

Prolog

Costrutti Prolog

Unificazione

Conjunctive queries

Le Regole

Ragionamento

Liste

Applicazioni

Programmazione
nondeterministica

In che consiste

Tic tac toe

Analisi del gioco

I/O elementare

GUI

Unicità di Prolog

■ Ricerca dei membri pari di una lista

```
/* stile generate and test */
membro_pari(X,L) :- member(X,L), 0 is X mod 2.

/* invece di ricorsione ad hoc */
membro_pari(X,[X|_]) :- 0 is X mod 2.
membro_pari(X,[_|Resto]) :- membro_pari(X,Resto).
```

■ Notare come l'approccio generate & test possa giocare il ruolo delle funzioni di ordine superiore in ML

- ◆ permette di comporre facilmente nuovi predicati da quelli dati
- ◆ in questo caso `member`, che diventa uno strumento generale per visitare una lista
- ◆ funge da *filter*: la query congiuntiva

```
member(X,Lista), predicato(X).
```

è l'analogo di: `filter predicato Lista`



Applicazione ai giochi

[Paradigma logico](#)

[Prolog](#)

[Costrutti Prolog](#)

[Unificazione](#)

[Conjunctive queries](#)

[Le Regole](#)

[Ragionamento](#)

[Liste](#)

[Applicazioni](#)

[Programmazione
nondeterministica](#)

[In che consiste](#)

Tic tac toe

Analisi del gioco

I/O elementare

GUI

[Unicità di Prolog](#)

■ Un possibile pattern generate-and-test per i giochi

```
next_move(Player ,Move) :-  
    possible_move(Player ,Move) , optimal(Player ,Move).
```



Applicazione ai giochi

[Paradigma logico](#)

[Prolog](#)

[Costrutti Prolog](#)

[Unificazione](#)

[Conjunctive queries](#)

[Le Regole](#)

[Ragionamento](#)

[Liste](#)

[Applicazioni](#)

[Programmazione
nondeterministica](#)

[In che consiste](#)

Tic tac toe

Analisi del gioco

I/O elementare

GUI

[Unicità di Prolog](#)

- Un possibile pattern generate-and-test per i giochi

```
next_move(Player ,Move) :-  
    possible_move(Player ,Move) , optimal(Player ,Move).
```

- A sua volta il controllo di ottimalità (cioè verificare se con Move sicuramente può vincere o almeno pareggiare) può essere effettuato con generate-and-test



Applicazione ai giochi

Paradigma logico

Prolog

Costrutti Prolog

Unificazione

Conjunctive queries

Le Regole

Ragionamento

Liste

Applicazioni

Programmazione
nondeterministica

In che consiste

Tic tac toe

Analisi del gioco

I/O elementare

GUI

Unicità di Prolog

- Un possibile pattern generate-and-test per i giochi

```
next_move(Player ,Move) :-  
    possible_move(Player ,Move), optimal(Player ,Move).
```

- A sua volta il controllo di ottimalità (cioè verificare se con Move sicuramente può vincere o almeno pareggiare) può essere effettuato con generate-and-test
- Nota: *optimal* mi dice se Move è la prima mossa di una strategia di gioco ottima → posso usare *optimal* per analizzare il gioco



Applicazione ai giochi

[Paradigma logico](#)

[Prolog](#)

[Costrutti Prolog](#)

[Unificazione](#)

[Conjunctive queries](#)

[Le Regole](#)

[Ragionamento](#)

[Liste](#)

[Applicazioni](#)

[Programmazione
nondeterministica](#)

[In che consiste](#)

Tic tac toe

Analisi del gioco

I/O elementare

GUI

[Unicità di Prolog](#)

- Un possibile pattern generate-and-test per i giochi

```
next_move(Player ,Move) :-  
    possible_move(Player ,Move), optimal(Player ,Move).
```

- A sua volta il controllo di ottimalità (cioè verificare se con Move sicuramente può vincere o almeno pareggiare) può essere effettuato con generate-and-test
- Nota: *optimal* mi dice se Move è la prima mossa di una strategia di gioco ottima → posso usare *optimal* per analizzare il gioco
 - ◆ ad es. il primo giocatore ha una strategia vincente?



Applicazione ai giochi

Paradigma logico

Prolog

Costrutti Prolog

Unificazione

Conjunctive queries

Le Regole

Ragionamento

Liste

Applicazioni

Programmazione
nondeterministica

In che consiste

Tic tac toe

Analisi del gioco

I/O elementare

GUI

Unicità di Prolog

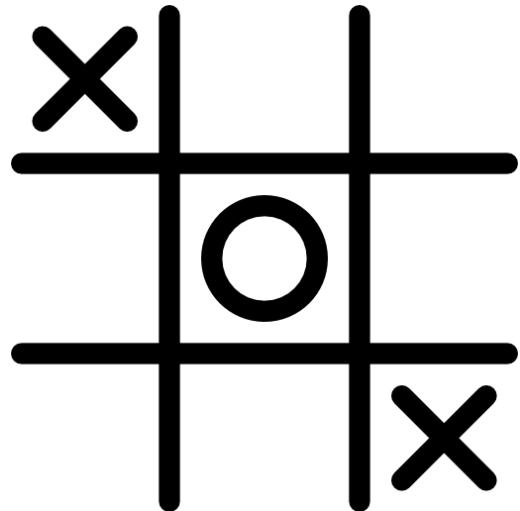
- Un possibile pattern generate-and-test per i giochi

```
next_move(Player ,Move) :-  
    possible_move(Player ,Move) , optimal(Player ,Move).
```

- A sua volta il controllo di ottimalità (cioè verificare se con Move sicuramente può vincere o almeno pareggiare) può essere effettuato con generate-and-test
- Nota: *optimal* mi dice se Move è la prima mossa di una strategia di gioco ottima → posso usare *optimal* per analizzare il gioco
 - ◆ ad es. il primo giocatore ha una strategia vincente?
 - ◆ mostrerà che è utile generare *tutte* le soluzioni



Caso di studio: Tic tac toe (aka tris)



Un semplice programma di AI che gioca a tris (imbattibile)

- una pagina di codice per “l’intelligenza”
- una pagina per i turni e l’interfaccia utente

[Paradigma logico](#)

[Prolog](#)

[Costrutti Prolog](#)

[Unificazione](#)

[Conjunctive queries](#)

[Le Regole](#)

[Ragionamento](#)

[Liste](#)

[Applicazioni](#)

[Programmazione nondeterministica](#)

In che consiste

Tic tac toe

Analisi del gioco

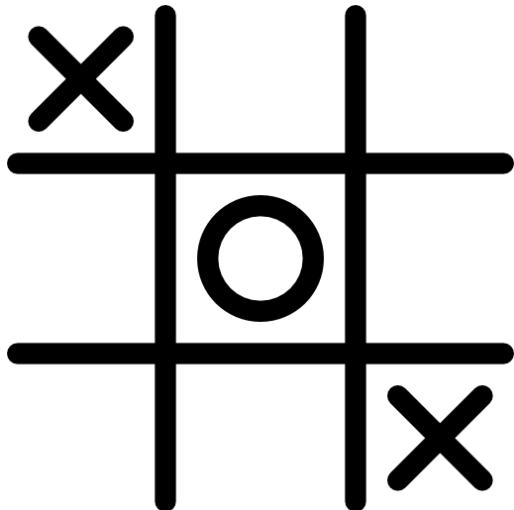
I/O elementare

GUI

[Unicità di Prolog](#)



Caso di studio: Tic tac toe (aka tris)



Un semplice programma di AI che gioca a tris (imbattibile)

- una pagina di codice per “l’intelligenza”
- una pagina per i turni e l’interfaccia utente

- Effettua una ricerca della strategia ottima:
 - ◆ ne adotta una vincente, se esiste
 - ◆ altrimenti mira al pareggio

[Paradigma logico](#)

[Prolog](#)

[Costrutti Prolog](#)

[Unificazione](#)

[Conjunctive queries](#)

[Le Regole](#)

[Ragionamento](#)

[Liste](#)

[Applicazioni](#)

[Programmazione nondeterministica](#)

In che consiste

Tic tac toe

Analisi del gioco

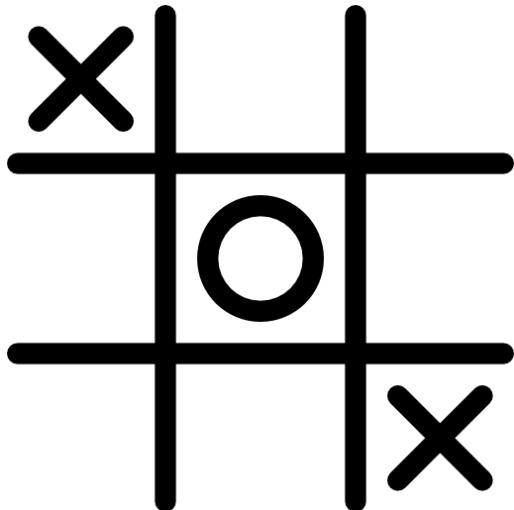
I/O elementare

GUI

[Unicità di Prolog](#)



Caso di studio: Tic tac toe (aka tris)



Un semplice programma di AI che gioca a tris (imbattibile)

- una pagina di codice per “l’intelligenza”
- una pagina per i turni e l’interfaccia utente

- Effettua una ricerca della strategia ottima:
 - ◆ ne adotta una vincente, se esiste
 - ◆ altrimenti mira al pareggio
- Approccio *generate-and-test* (programmazione nondeterministica) alla scelta della mossa. Ad ogni turno:
 1. genera una mossa
 2. verifica se è ottimale (la prima di una strategia ottima)



Tic tac toe – Descrizione del gioco

[Paradigma logico](#)

[Prolog](#)

[Costrutti Prolog](#)

[Unificazione](#)

[Conjunctive queries](#)

[Le Regole](#)

[Ragionamento](#)

[Liste](#)

[Applicazioni](#)

[Programmazione
nondeterministica](#)

In che consiste

Tic tac toe

Analisi del gioco

I/O elementare

GUI

[Unicità di Prolog](#)

Rappresentiamo la scacchiera come una lista di 3 righe (liste):

```
start([[1,2,3],[4,5,6],[7,8,9]]).
```



Tic tac toe – Descrizione del gioco

[Paradigma logico](#)

[Prolog](#)

[Costrutti Prolog](#)

[Unificazione](#)

[Conjunctive queries](#)

[Le Regole](#)

[Ragionamento](#)

[Liste](#)

[Applicazioni](#)

[Programmazione
nondeterministica](#)

In che consiste

Tic tac toe

Analisi del gioco

I/O elementare

GUI

[Unicità di Prolog](#)

Rappresentiamo la scacchiera come una lista di 3 righe (liste):

```
start([[1,2,3],[4,5,6],[7,8,9]]).
```

Introduciamo due simboli “x” e “o” per i giocatori. Quando un giocatore fa una mossa, il suo simbolo occuperà quella casella sulla scacchiera.

```
adversary(x,o).  
adversary(o,x).
```



Tic tac toe – Descrizione del gioco

[Paradigma logico](#)

[Prolog](#)

[Costrutti Prolog](#)

[Unificazione](#)

[Conjunctive queries](#)

[Le Regole](#)

[Ragionamento](#)

[Liste](#)

[Applicazioni](#)

[Programmazione
nondeterministica](#)

In che consiste

Tic tac toe

Analisi del gioco

I/O elementare

GUI

[Unicità di Prolog](#)

Rappresentiamo la scacchiera come una lista di 3 righe (liste):

```
start([[1,2,3],[4,5,6],[7,8,9]]).
```

Introduciamo due simboli “x” e “o” per i giocatori. Quando un giocatore fa una mossa, il suo simbolo occuperà quella casella sulla scacchiera.

```
adversary(x,o).  
adversary(o,x).
```

Definiamo la condizione di vittoria:

```
win(P, [[P,P,P],_,_]).  
win(P, [_,[P,P,P],_]).  
win(P, [_,_,[P,P,P]]).
```



Tic tac toe – Descrizione del gioco

Paradigma logico

Prolog

Costrutti Prolog

Unificazione

Conjunctive queries

Le Regole

Ragionamento

Liste

Applicazioni

Programmazione
nondeterministica

In che consiste

Tic tac toe

Analisi del gioco

I/O elementare

GUI

Unicità di Prolog

Rappresentiamo la scacchiera come una lista di 3 righe (liste):

```
start([[1,2,3],[4,5,6],[7,8,9]]).
```

Introduciamo due simboli “x” e “o” per i giocatori. Quando un giocatore fa una mossa, il suo simbolo occuperà quella casella sulla scacchiera.

```
adversary(x,o).  
adversary(o,x).
```

Definiamo la condizione di vittoria:

```
win(P, [[P,P,P],_,_]).  
win(P, [_,[P,P,P],_]).  
win(P, [_,_,[P,P,P]]).  
win(P, [[P,_,_],[P,_,_],[P,_,_]]).  
win(P, [[_,P,_],[_,P,_],[_,P,_]]).  
win(P, [[_,_,P],[_,_,P],[_,_,P]])
```



Tic tac toe – Descrizione del gioco

[Paradigma logico](#)

[Prolog](#)

[Costrutti Prolog](#)

[Unificazione](#)

[Conjunctive queries](#)

[Le Regole](#)

[Ragionamento](#)

[Liste](#)

[Applicazioni](#)

[Programmazione
nondeterministica](#)

In che consiste

Tic tac toe

Analisi del gioco

I/O elementare

GUI

[Unicità di Prolog](#)

Rappresentiamo la scacchiera come una lista di 3 righe (liste):

```
start([[1,2,3],[4,5,6],[7,8,9]]).
```

Introduciamo due simboli “x” e “o” per i giocatori. Quando un giocatore fa una mossa, il suo simbolo occuperà quella casella sulla scacchiera.

```
adversary(x,o).  
adversary(o,x).
```

Definiamo la condizione di vittoria:

```
win(P, [[P,P,P],_,_]).  
win(P, [_,[P,P,P],_]).  
win(P, [_,_,[P,P,P]]).  
win(P, [[P,_,_],[P,_,_],[P,_,_]]).  
win(P, [[_,P,_],[_,P,_],[_,P,_]]).  
win(P, [[_,_,P],[_,_,P],[_,_,P]]).  
win(P, [[P,_,_],[_,P,_],[_,_,P]]).  
win(P, [[_,_,P],[_,P,_],[P,_,_]]).
```



Tic tac toe – Descrizione del gioco

[Paradigma logico](#)

[Prolog](#)

[Costrutti Prolog](#)

[Unificazione](#)

[Conjunctive queries](#)

[Le Regole](#)

[Ragionamento](#)

[Liste](#)

[Applicazioni](#)

[Programmazione
nondeterministica](#)

In che consiste

Tic tac toe

Analisi del gioco

I/O elementare

GUI

[Unicità di Prolog](#)

Definiamo una situazione “non finale” (nessuno ha ancora vinto e la scacchiera non è piena):

```
non_final(Board) :-  
    \+ win(_,Board),  
    member(Row,Board),  
    member(Cell,Row),  
    number(Cell).      /* predicato built-in */
```



Tic tac toe – Descrizione del gioco

[Paradigma logico](#)

[Prolog](#)

[Costrutti Prolog](#)

[Unificazione](#)

[Conjunctive queries](#)

[Le Regole](#)

[Ragionamento](#)

[Liste](#)

[Applicazioni](#)

[Programmazione
nondeterministica](#)

[In che consiste](#)

Tic tac toe

[Analisi del gioco](#)

[I/O elementare](#)

[GUI](#)

[Unicità di Prolog](#)

Definiamo una situazione “non finale” (nessuno ha ancora vinto e la scacchiera non è piena):

```
non_final(Board) :-  
    \+ win(_,Board),  
    member(Row,Board),  
    member(Cell,Row),  
    number(Cell).      /* predicato built-in */  
  
final(Board) :-  
    \+ non_final(Board).
```



Tic tac toe – Possibili mosse e loro effetto

[Paradigma logico](#)

[Prolog](#)

[Costrutti Prolog](#)

[Unificazione](#)

[Conjunctive queries](#)

[Le Regole](#)

[Ragionamento](#)

[Liste](#)

[Applicazioni](#)

[Programmazione
nondeterministica](#)

In che consiste

Tic tac toe

Analisi del gioco

I/O elementare

GUI

[Unicità di Prolog](#)

Schema: `move(+P,+N,+Board1,-Board2)`



Tic tac toe – Possibili mosse e loro effetto

[Paradigma logico](#)

[Prolog](#)

[Costrutti Prolog](#)

[Unificazione](#)

[Conjunctive queries](#)

[Le Regole](#)

[Ragionamento](#)

[Liste](#)

[Applicazioni](#)

[Programmazione
nondeterministica](#)

In che consiste

Tic tac toe

Analisi del gioco

I/O elementare

GUI

[Unicità di Prolog](#)

Schema: `move(+P,+N,+Board1,-Board2)` dove

- **P** (player) è il simbolo del giocatore che fa la mossa ('x' oppure 'o')



Tic tac toe – Possibili mosse e loro effetto

[Paradigma logico](#)

[Prolog](#)

[Costrutti Prolog](#)

[Unificazione](#)

[Conjunctive queries](#)

[Le Regole](#)

[Ragionamento](#)

[Liste](#)

[Applicazioni](#)

[Programmazione
nondeterministica](#)

In che consiste

Tic tac toe

Analisi del gioco

I/O elementare

GUI

[Unicità di Prolog](#)

Schema: `move(+P,+N,+Board1,-Board2)` dove

- **P** (player) è il simbolo del giocatore che fa la mossa ('x' oppure 'o')
- **N** la mossa, indicata dal numero della cella ($1 \leq N \leq 9$)



Tic tac toe – Possibili mosse e loro effetto

[Paradigma logico](#)

[Prolog](#)

[Costrutti Prolog](#)

[Unificazione](#)

[Conjunctive queries](#)

[Le Regole](#)

[Ragionamento](#)

[Liste](#)

[Applicazioni](#)

[Programmazione
nondeterministica](#)

In che consiste

Tic tac toe

Analisi del gioco

I/O elementare

GUI

[Unicità di Prolog](#)

Schema: `move(+P,+N,+Board1,-Board2)` dove

- `P` (player) è il simbolo del giocatore che fa la mossa ('x' oppure 'o')
- `N` la mossa, indicata dal numero della cella ($1 \leq N \leq 9$)
- `Board1` è l'attuale scacchiera
- `Board2` è la scacchiera dopo la mossa



Tic tac toe – Possibili mosse e loro effetto

[Paradigma logico](#)

[Prolog](#)

[Costrutti Prolog](#)

[Unificazione](#)

[Conjunctive queries](#)

[Le Regole](#)

[Ragionamento](#)

[Liste](#)

[Applicazioni](#)

[Programmazione
nondeterministica](#)

In che consiste

Tic tac toe

Analisi del gioco

I/O elementare

GUI

[Unicità di Prolog](#)

Schema: `move(+P,+N,+Board1,-Board2)` dove

- `P` (player) è il simbolo del giocatore che fa la mossa ('x' oppure 'o')
- `N` la mossa, indicata dal numero della cella ($1 \leq N \leq 9$)
- `Board1` è l'attuale scacchiera
- `Board2` è la scacchiera dopo la mossa
- Convenzione nella documentazione Prolog: `+ = IN`, `- = OUT`, `niente = IN-OUT` (solo un suggerimento al programmatore, data l'invertibilità)



Tic tac toe – Possibili mosse e loro effetto

[Paradigma logico](#)

[Prolog](#)

[Costrutti Prolog](#)

[Unificazione](#)

[Conjunctive queries](#)

[Le Regole](#)

[Ragionamento](#)

[Liste](#)

[Applicazioni](#)

[Programmazione
nondeterministica](#)

In che consiste

Tic tac toe

Analisi del gioco

I/O elementare

GUI

[Unicità di Prolog](#)

Schema: `move(+P,+N,+Board1,-Board2)` dove

- `P` (player) è il simbolo del giocatore che fa la mossa ('x' oppure 'o')
- `N` la mossa, indicata dal numero della cella ($1 \leq N \leq 9$)
- `Board1` è l'attuale scacchiera
- `Board2` è la scacchiera dopo la mossa
- Convenzione nella documentazione Prolog: `+ = IN`, `- = OUT`, `niente = IN-OUT` (solo un suggerimento al programmatore, data l'invertibilità)

```
move(P, N, Board1, Board2) :-  
    \+ win(_, Board1),
```



Tic tac toe – Possibili mosse e loro effetto

[Paradigma logico](#)

[Prolog](#)

[Costrutti Prolog](#)

[Unificazione](#)

[Conjunctive queries](#)

[Le Regole](#)

[Ragionamento](#)

[Liste](#)

[Applicazioni](#)

[Programmazione
nondeterministica](#)

[In che consiste](#)

Tic tac toe

[Analisi del gioco](#)

[I/O elementare](#)

[GUI](#)

[Unicità di Prolog](#)

Schema: `move(+P,+N,+Board1,-Board2)` dove

- `P` (player) è il simbolo del giocatore che fa la mossa ('x' oppure 'o')
- `N` la mossa, indicata dal numero della cella ($1 \leq N \leq 9$)
- `Board1` è l'attuale scacchiera
- `Board2` è la scacchiera dopo la mossa
- Convenzione nella documentazione Prolog: `+ = IN`, `- = OUT`, `niente = IN-OUT` (solo un suggerimento al programmatore, data l'invertibilità)

```
move(P, N, Board1, Board2) :-  
    \+ win(_, Board1),  
    append(RowsBefore, [Row | RowsAfter], Board1),  
    append(CellsBefore, [N | CellsAfter], Row),
```



Tic tac toe – Possibili mosse e loro effetto

[Paradigma logico](#)

[Prolog](#)

[Costrutti Prolog](#)

[Unificazione](#)

[Conjunctive queries](#)

[Le Regole](#)

[Ragionamento](#)

[Liste](#)

[Applicazioni](#)

[Programmazione
nondeterministica](#)

In che consiste

Tic tac toe

Analisi del gioco

I/O elementare

GUI

[Unicità di Prolog](#)

Schema: `move(+P,+N,+Board1,-Board2)` dove

- `P` (player) è il simbolo del giocatore che fa la mossa ('x' oppure 'o')
- `N` la mossa, indicata dal numero della cella ($1 \leq N \leq 9$)
- `Board1` è l'attuale scacchiera
- `Board2` è la scacchiera dopo la mossa
- Convenzione nella documentazione Prolog: `+ = IN`, `- = OUT`, `niente = IN-OUT` (solo un suggerimento al programmatore, data l'invertibilità)

```
move(P, N, Board1, Board2) :-  
  \+ win(_, Board1),  
  append(RowsBefore, [Row | RowsAfter], Board1),  
  append(CellsBefore, [N | CellsAfter], Row),  
  number(N),
```



Tic tac toe – Possibili mosse e loro effetto

[Paradigma logico](#)

[Prolog](#)

[Costrutti Prolog](#)

[Unificazione](#)

[Conjunctive queries](#)

[Le Regole](#)

[Ragionamento](#)

[Liste](#)

[Applicazioni](#)

[Programmazione
nondeterministica](#)

In che consiste

Tic tac toe

Analisi del gioco

I/O elementare

GUI

[Unicità di Prolog](#)

Schema: `move(+P,+N,+Board1,-Board2)` dove

- `P` (player) è il simbolo del giocatore che fa la mossa ('x' oppure 'o')
- `N` la mossa, indicata dal numero della cella ($1 \leq N \leq 9$)
- `Board1` è l'attuale scacchiera
- `Board2` è la scacchiera dopo la mossa
- Convenzione nella documentazione Prolog: `+ = IN`, `- = OUT`, `niente = IN-OUT` (solo un suggerimento al programmatore, data l'invertibilità)

```
move(P, N, Board1, Board2) :-  
  \+ win(_, Board1),  
  append(RowsBefore, [Row | RowsAfter], Board1),  
  append(CellsBefore, [N | CellsAfter], Row),  
  number(N),  
  append(CellsBefore, [P | CellsAfter], NewRow),  
  append(RowsBefore, [NewRow | RowsAfter], Board2).
```



Tic tac toe – Le strategie (generate and test)

[Paradigma logico](#)

[Prolog](#)

[Costrutti Prolog](#)

[Unificazione](#)

[Conjunctive queries](#)

[Le Regole](#)

[Ragionamento](#)

[Liste](#)

[Applicazioni](#)

[Programmazione
nondeterministica](#)

In che consiste

Tic tac toe

Analisi del gioco

I/O elementare

GUI

[Unicità di Prolog](#)

Schema: `has_XXX_strat(+P,-Move,+Board)` dove

- `P` (player) è il simbolo ‘x’ oppure ‘o’
- `Move` è la prima mossa della strategia
- `Board` è l’attuale scacchiera

Significato: *nella situazione descritta da Board, P ha una strategia di tipo XXX (vincente o non-perdente) che inizia con Move*



Tic tac toe – Le strategie (generate and test)

[Paradigma logico](#)

[Prolog](#)

[Costrutti Prolog](#)

[Unificazione](#)

[Conjunctive queries](#)

[Le Regole](#)

[Ragionamento](#)

[Liste](#)

[Applicazioni](#)

[Programmazione
nondeterministica](#)

[In che consiste](#)

Tic tac toe

[Analisi del gioco](#)

[I/O elementare](#)

[GUI](#)

[Unicità di Prolog](#)

Schema: `has_XXX_strat(+P,-Move,+Board)` dove

- `P` (player) è il simbolo ‘x’ oppure ‘o’
- `Move` è la prima mossa della strategia
- `Board` è l’attuale scacchiera

Significato: *nella situazione descritta da Board, P ha una strategia di tipo XXX (vincente o non-perdente) che inizia con Move*

```
has_win_strat(P,_,Board) :-  
    win(P,Board).  
has_win_strat(P,Move,Board) :-  
    move(P,Move,Board,Board2),  
    adversary(P,Adv),  
    \+ has_tie_strat(Adv,_,Board2).
```



Tic tac toe – Le strategie (generate and test)

Paradigma logico

Prolog

Costrutti Prolog

Unificazione

Conjunctive queries

Le Regole

Ragionamento

Liste

Applicazioni

Programmazione
nondeterministica

In che consiste

Tic tac toe

Analisi del gioco

I/O elementare

GUI

Unicità di Prolog

Schema: `has_XXX_strat(+P,-Move,+Board)` dove

- `P` (player) è il simbolo ‘x’ oppure ‘o’
- `Move` è la prima mossa della strategia
- `Board` è l’attuale scacchiera

Significato: *nella situazione descritta da Board, P ha una strategia di tipo XXX (vincente o non-perdente) che inizia con Move*

```
has_win_strat(P,_,Board) :-  
    win(P,Board).  
  
has_win_strat(P,Move,Board) :-  
    move(P,Move,Board,Board2),  
    adversary(P,Adv),  
    \+ has_tie_strat(Adv,_,Board2).  
  
has_tie_strat( _,_,Board) :-  
    final(Board),  
    \+ win(_,Board).  
  
has_tie_strat(P,Move,Board) :-  
    move(P,Move,Board,Board2),  
    adversary(P,Adv),  
    \+ has_win_strat(Adv,_,Board2).
```



Tic-tac-toe – Analisi del gioco

- Esiste una strategia vincente per il primo giocatore?

[Paradigma logico](#)

[Prolog](#)

[Costrutti Prolog](#)

[Unificazione](#)

[Conjunctive queries](#)

[Le Regole](#)

[Ragionamento](#)

[Liste](#)

[Applicazioni](#)

[Programmazione
nondeterministica](#)

In che consiste

Tic tac toe

[**Analisi del gioco**](#)

I/O elementare

GUI

[Unicità di Prolog](#)



Tic-tac-toe – Analisi del gioco

Paradigma logico

Prolog

Costrutti Prolog

Unificazione

Conjunctive queries

Le Regole

Ragionamento

Liste

Applicazioni

Programmazione
nondeterministica

In che consiste

Tic tac toe

Analisi del gioco

I/O elementare

GUI

Unicità di Prolog

- Esiste una strategia vincente per il primo giocatore?

```
?- start(Board), has_win_strat(x, Move, Board).  
false.
```

NO: nessuna mossa iniziale garantisce al primo giocatore di vincere



Tic-tac-toe – Analisi del gioco

Paradigma logico

Prolog

Costrutti Prolog

Unificazione

Conjunctive queries

Le Regole

Ragionamento

Liste

Applicazioni

Programmazione
nondeterministica

In che consiste

Tic tac toe

Analisi del gioco

I/O elementare

GUI

Unicità di Prolog

- Esiste una strategia vincente per il primo giocatore?

```
?- start(Board), has_win_strat(x, Move, Board).  
false.
```

NO: nessuna mossa iniziale garantisce al primo giocatore di vincere

- Esiste una strategia vincente per il secondo giocatore?



Tic-tac-toe – Analisi del gioco

Paradigma logico

Prolog

Costrutti Prolog

Unificazione

Conjunctive queries

Le Regole

Ragionamento

Liste

Applicazioni

Programmazione nondeterministica

In che consiste

Tic tac toe

Analisi del gioco

I/O elementare

GUI

Unicità di Prolog

■ Esiste una strategia vincente per il primo giocatore?

```
?- start(Board), has_win_strat(x, Move, Board).  
false.
```

NO: nessuna mossa iniziale garantisce al primo giocatore di vincere

■ Esiste una strategia vincente per il secondo giocatore?

```
?- start(B0), has_tie_strat(x, Mv, B0).  
B0 = [[1, 2, 3], [4, 5, 6], [7, 8, 9]],  
Mv = 1 ;  
...  
B0 = [[1, 2, 3], [4, 5, 6], [7, 8, 9]],  
Mv = 9 ;  
false.
```

NO: ogni mossa iniziale permette al primo giocatore di pareggiare (se non commette errori)

- ◆ tutte le mosse iniziali fanno parte di una strategia di pareggio



Tic-tac-toe – Analisi del gioco (II)

■ Come può pareggiare il secondo giocatore?

[Paradigma logico](#)

[Prolog](#)

[Costrutti Prolog](#)

[Unificazione](#)

[Conjunctive queries](#)

[Le Regole](#)

[Ragionamento](#)

[Liste](#)

[Applicazioni](#)

[Programmazione
nondeterministica](#)

In che consiste

Tic tac toe

[**Analisi del gioco**](#)

I/O elementare

GUI

[Unicità di Prolog](#)



Tic-tac-toe – Analisi del gioco (II)

Paradigma logico

Prolog

Costrutti Prolog

Unificazione

Conjunctive queries

Le Regole

Ragionamento

Liste

Applicazioni

Programmazione
nondeterministica

In che consiste

Tic tac toe

Analisi del gioco

I/O elementare

GUI

Unicità di Prolog

■ Come può pareggiare il secondo giocatore?

```
?- start(B0), move(x,1,B0,B1), has_tie_strat(o,Mv,B1).  
B0 = [[1, 2, 3], [4, 5, 6], [7, 8, 9]],  
B1 = [[x, 2, 3], [4, 5, 6], [7, 8, 9]],  
Mv = 5 ;  
false.
```

Se la prima mossa è su un angolo, l'unica risposta è 5, in tutti gli altri casi vince il primo giocatore



Tic-tac-toe – Analisi del gioco (II)

Paradigma logico

Prolog

Costrutti Prolog

Unificazione

Conjunctive queries

Le Regole

Ragionamento

Liste

Applicazioni

Programmazione
nondeterministica

In che consiste

Tic tac toe

Analisi del gioco

I/O elementare

GUI

Unicità di Prolog

■ Come può pareggiare il secondo giocatore?

```
?- start(B0), move(x,1,B0,B1), has_tie_strat(o,Mv,B1).  
B0 = [[1, 2, 3], [4, 5, 6], [7, 8, 9]],  
B1 = [[x, 2, 3], [4, 5, 6], [7, 8, 9]],  
Mv = 5 ;  
false.
```

Se la prima mossa è su un angolo, l'unica risposta è 5, in tutti gli altri casi vince il primo giocatore

```
?- start(B0), move(x,2,B0,B1), has_tie_strat(o,Mv,B1).  
...  
Mv = 1 ;  
...  
Mv = 3 ;  
...  
Mv = 5 ;  
...  
Mv = 8 ;  
false.
```

Quindi se la prima mossa è 2, meglio non rispondere con 4, 6, 7 o 9!



Tic-tac-toe – Analisi del gioco (III)

[Paradigma logico](#)

[Prolog](#)

[Costrutti Prolog](#)

[Unificazione](#)

[Conjunctive queries](#)

[Le Regole](#)

[Ragionamento](#)

[Liste](#)

[Applicazioni](#)

[Programmazione
nondeterministica](#)

In che consiste

Tic tac toe

[Analisi del gioco](#)

I/O elementare

GUI

[Unicità di Prolog](#)

■ Come può pareggiare il secondo giocatore?

```
?- start(B0), move(x,5,B0,B1), has_tie_strat(o,Mv,B1).
```



Tic-tac-toe – Analisi del gioco (III)

Paradigma logico

Prolog

Costrutti Prolog

Unificazione

Conjunctive queries

Le Regole

Ragionamento

Liste

Applicazioni

Programmazione
nondeterministica

In che consiste

Tic tac toe

Analisi del gioco

I/O elementare

GUI

Unicità di Prolog

- Come può pareggiare il secondo giocatore?

```
?- start(B0), move(x,5,B0,B1), has_tie_strat(o,Mv,B1).  
...  
Mv = 1 ;  
...  
Mv = 3 ;  
...  
Mv = 7 ;  
...  
Mv = 9 ;  
false.
```

Se la prima mossa è 5, si deve scegliere una casella d'angolo

- Tutte queste affermazioni si possono verificare empiricamente forzando il primo passo:

```
?- start(B0), move(x,5,B0,B1), turn(user,o,B1).
```



Tic tac toe – L'interfaccia utente testuale

[Paradigma logico](#)

[Prolog](#)

[Costrutti Prolog](#)

[Unificazione](#)

[Conjunctive queries](#)

[Le Regole](#)

[Ragionamento](#)

[Liste](#)

[Applicazioni](#)

[Programmazione
nondeterministica](#)

In che consiste

Tic tac toe

Analisi del gioco

I/O elementare

GUI

[Unicità di Prolog](#)

(per le interfacce grafiche vedere l'estensione XPCE,
<https://www.swi-prolog.org/packages/xpce/>)

```
cpu move: 1
x|2|3
4|5|6
7|8|9

Player o insert your move [1-9]: 8
x|2|3
4|5|6
7|o|9
```



Tic tac toe – L'interfaccia utente testuale

[Paradigma logico](#)

[Prolog](#)

[Costrutti Prolog](#)

[Unificazione](#)

[Conjunctive queries](#)

[Le Regole](#)

[Ragionamento](#)

[Liste](#)

[Applicazioni](#)

[Programmazione
nondeterministica](#)

In che consiste

Tic tac toe

Analisi del gioco

I/O elementare

GUI

[Unicità di Prolog](#)

(per le interfacce grafiche vedere l'estensione XPCE,
<https://www.swi-prolog.org/packages/xpce/>)

```
cpu move: 1
x|2|3
4|5|6
7|8|9

Player o insert your move [1-9]: 8
x|2|3
4|5|6
7|o|9
```

```
print_board([]).
print_board([Row|Rest]) :-
    format('~a|~a|~a\n', Row),
    print_board(Rest).
```



Tic tac toe – L'interfaccia utente testuale

[Paradigma logico](#)

[Prolog](#)

[Costrutti Prolog](#)

[Unificazione](#)

[Conjunctive queries](#)

[Le Regole](#)

[Ragionamento](#)

[Liste](#)

[Applicazioni](#)

[Programmazione
nondeterministica](#)

In che consiste

Tic tac toe

Analisi del gioco

I/O elementare

GUI

[Unicità di Prolog](#)

(per le interfacce grafiche vedere l'estensione XPCE,
<https://www.swi-prolog.org/packages/xpce/>)

```
cpu move: 1
x|2|3
4|5|6
7|8|9

Player o insert your move [1-9]: 8
x|2|3
4|5|6
7|o|9
```

```
print_board([]).
print_board([Row|Rest]) :-
    format('`~a|`~a|`~a`\n', Row),
    print_board(Rest).

read_move(Player,Move) :-
    format('\nPlayer `~a insert your move [1-9]: ', [Player]),
    get_single_char(Char), put_char(Char), nl,
    Move is Char-48,
    Move >= 1, Move =< 9.
```



Tic tac toe – I turni

[Paradigma logico](#)

[Prolog](#)

[Costrutti Prolog](#)

[Unificazione](#)

[Conjunctive queries](#)

[Le Regole](#)

[Ragionamento](#)

[Liste](#)

[Applicazioni](#)

[Programmazione
nondeterministica](#)

In che consiste

Tic tac toe

Analisi del gioco

I/O elementare

GUI

[Unicità di Prolog](#)

```
turn(_,_,Board) :-  
    final(Board),  
    \+ win(_,Board),  
    format('`The game ends in a tie.\n').
```



Tic tac toe – I turni

[Paradigma logico](#)

[Prolog](#)

[Costrutti Prolog](#)

[Unificazione](#)

[Conjunctive queries](#)

[Le Regole](#)

[Ragionamento](#)

[Liste](#)

[Applicazioni](#)

[Programmazione
nondeterministica](#)

In che consiste

Tic tac toe

Analisi del gioco

I/O elementare

GUI

[Unicità di Prolog](#)

```
turn(_,_,Board) :-  
    final(Board),  
    \+ win(_,Board),  
    format('`The game ends in a tie.\n').
```

```
turn(P,_,Board) :-  
    win(_,Board),  
    member(Adv,[user,cpu]), Adv \= P,  
    format(`The ~a wins!\n', [Adv]).
```



Tic tac toe – I turni

[Paradigma logico](#)

[Prolog](#)

[Costrutti Prolog](#)

[Unificazione](#)

[Conjunctive queries](#)

[Le Regole](#)

[Ragionamento](#)

[Liste](#)

[Applicazioni](#)

[Programmazione
nondeterministica](#)

In che consiste

Tic tac toe

Analisi del gioco

I/O elementare

GUI

[Unicità di Prolog](#)

```
turn(_,_,Board) :-  
    final(Board),  
    \+ win(_,Board),  
    format('`The game ends in a tie.\n').  
  
turn(P,_,Board) :-  
    win(_,Board),  
    member(Adv,[user,cpu]), Adv \= P,  
    format(`The ~a wins!\n', [Adv]).  
  
turn(user,P,Board) :-  
    read_move(P,M),  
    move(P, M, Board, Board2),  
    print_board(Board2),  
    adversary(P,Adv),  
    turn(cpu,Adv,Board2).
```



Tic tac toe – I turni

[Paradigma logico](#)

[Prolog](#)

[Costrutti Prolog](#)

[Unificazione](#)

[Conjunctive queries](#)

[Le Regole](#)

[Ragionamento](#)

[Liste](#)

[Applicazioni](#)

[Programmazione
nondeterministica](#)

In che consiste

Tic tac toe

Analisi del gioco

I/O elementare

GUI

[Unicità di Prolog](#)

```
turn(_,_,Board) :-  
    final(Board),  
    \+ win(_,Board),  
    format('`The game ends in a tie.\n').  
  
turn(P,_,Board) :-  
    win(_,Board),  
    member(Adv,[user,cpu]), Adv \= P,  
    format(`The ~a wins!\n', [Adv]).  
  
turn(user,P,Board) :-  
    read_move(P,M),  
    move(P, M, Board, Board2),  
    print_board(Board2),  
    adversary(P,Adv),  
    turn(cpu,Adv,Board2).  
  
turn(cpu,P,Board) :-  
    (has_win_strat(P,Move,Board); has_tie_strat(P,Move,Board)),  
    format(`cpu move: ~a\n', [Move]),  
    move(P, Move, Board, Board2),  
    print_board(Board2),  
    adversary(P,Adv),  
    turn(user,Adv,Board2).
```



Tic-tac-toe – II “main”

Due modalità di esecuzione:

- Specificando liberamente chi inizia e che simbolo usa

[Paradigma logico](#)

[Prolog](#)

[Costrutti Prolog](#)

[Unificazione](#)

[Conjunctive queries](#)

[Le Regole](#)

[Ragionamento](#)

[Liste](#)

[Applicazioni](#)

[Programmazione
nondeterministica](#)

In che consiste

Tic tac toe

Analisi del gioco

I/O elementare

GUI

[Unicità di Prolog](#)



Tic-tac-toe – II “main”

Due modalità di esecuzione:

- Specificando liberamente chi inizia e che simbolo usa

```
go(First ,Symbol) :-  
    member(First ,[user ,cpu]) ,  
    start(Board) ,  
    turn(First ,Symbol ,Board).
```

[Paradigma logico](#)

[Prolog](#)

[Costrutti Prolog](#)

[Unificazione](#)

[Conjunctive queries](#)

[Le Regole](#)

[Ragionamento](#)

[Liste](#)

[Applicazioni](#)

[Programmazione
nondeterministica](#)

In che consiste

Tic tac toe

Analisi del gioco

I/O elementare

GUI

[Unicità di Prolog](#)



Tic-tac-toe – II “main”

Due modalità di esecuzione:

- Specificando liberamente chi inizia e che simbolo usa

```
go(First ,Symbol) :-  
    member(First ,[user ,cpu]) ,  
    start(Board) ,  
    turn(First ,Symbol ,Board).
```

Esempio di invocazione

```
?- go(user ,x).
```

Paradigma logico

Prolog

Costrutti Prolog

Unificazione

Conjunctive queries

Le Regole

Ragionamento

Liste

Applicazioni

Programmazione
nondeterministica

In che consiste

Tic tac toe

Analisi del gioco

I/O elementare

GUI

Unicità di Prolog



Tic-tac-toe – II “main”

Paradigma logico

Prolog

Costrutti Prolog

Unificazione

Conjunctive queries

Le Regole

Ragionamento

Liste

Applicazioni

Programmazione
nondeterministica

In che consiste

Tic tac toe

Analisi del gioco

I/O elementare

GUI

Unicità di Prolog

Due modalità di esecuzione:

- Specificando liberamente chi inizia e che simbolo usa

```
go(First,Symbol) :-  
    member(First,[user,cpu]),  
    start(Board),  
    turn(First,Symbol,Board).
```

Esempio di invocazione

```
?- go(user,x).
```

- Estrazione a sorte di chi inizia

```
main :-  
    choice is random(2),  
    nth0(choice,[user,o],(cpu,x)),(First,Symbol)),  
    format('The ~a starts\n', [First]),  
    start(Board),  
    turn(First,Symbol,Board),  
    halt.
```

Demo



Tic-tac-toe – Compilazione in codice stand-alone

[Paradigma logico](#)

[Prolog](#)

[Costrutti Prolog](#)

[Unificazione](#)

[Conjunctive queries](#)

[Le Regole](#)

[Ragionamento](#)

[Liste](#)

[Applicazioni](#)

[Programmazione
nondeterministica](#)

In che consiste

Tic tac toe

Analisi del gioco

I/O elementare

GUI

[Unicità di Prolog](#)

```
swipl --goal=main --stand_alone=true -o tic-tac-toe  
      -c tic-tac-toe.pl
```

■ Spiegazione delle opzioni:

- ◆ **--stand_alone**: crea un codice direttamente eseguibile (invece di far partire la macchina virtuale)



Tic-tac-toe – Compilazione in codice stand-alone

[Paradigma logico](#)

[Prolog](#)

[Costrutti Prolog](#)

[Unificazione](#)

[Conjunctive queries](#)

[Le Regole](#)

[Ragionamento](#)

[Liste](#)

[Applicazioni](#)

[Programmazione
nondeterministica](#)

In che consiste

Tic tac toe

Analisi del gioco

I/O elementare

GUI

[Unicità di Prolog](#)

```
swipl --goal=main --stand_alone=true -o tic-tac-toe  
      -c tic-tac-toe.pl
```

■ Spiegazione delle opzioni:

- ◆ --stand_alone: crea un codice direttamente eseguibile (invece di far partire la macchina virtuale)
- ◆ -o: nome del file oggetto



Tic-tac-toe – Compilazione in codice stand-alone

[Paradigma logico](#)

[Prolog](#)

[Costrutti Prolog](#)

[Unificazione](#)

[Conjunctive queries](#)

[Le Regole](#)

[Ragionamento](#)

[Liste](#)

[Applicazioni](#)

[Programmazione
nondeterministica](#)

In che consiste

Tic tac toe

Analisi del gioco

I/O elementare

GUI

[Unicità di Prolog](#)

```
swipl --goal=main --stand_alone=true -o tic-tac-toe  
      -c tic-tac-toe.pl
```

■ Spiegazione delle opzioni:

- ◆ --stand_alone: crea un codice direttamente eseguibile (invece di far partire la macchina virtuale)
- ◆ -o: nome del file oggetto
- ◆ -c: nome del file sorgente



Tic-tac-toe – Compilazione in codice stand-alone

[Paradigma logico](#)

[Prolog](#)

[Costrutti Prolog](#)

[Unificazione](#)

[Conjunctive queries](#)

[Le Regole](#)

[Ragionamento](#)

[Liste](#)

[Applicazioni](#)

[Programmazione
nondeterministica](#)

In che consiste

Tic tac toe

Analisi del gioco

I/O elementare

GUI

[Unicità di Prolog](#)

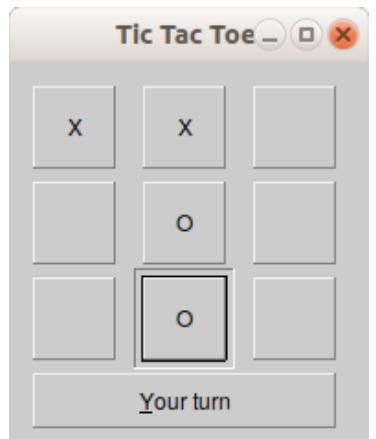
```
swipl --goal=main --stand_alone=true -o tic-tac-toe  
      -c tic-tac-toe.pl
```

■ Spiegazione delle opzioni:

- ◆ --stand_alone: crea un codice direttamente eseguibile (invece di far partire la macchina virtuale)
- ◆ -o: nome del file oggetto
- ◆ -c: nome del file sorgente
- ◆ --goal: il goal da invocare quando il file viene eseguito

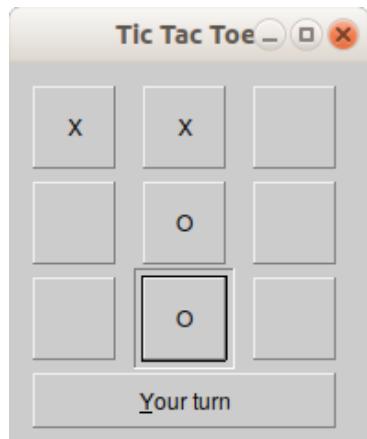


Cenni a una possibile interfaccia grafica con XPCE





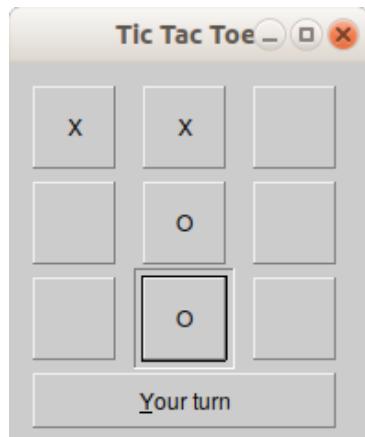
Cenni a una possibile interfaccia grafica con XPCE



```
gui :-  
  /* costruzione buttoni */  
  
  new(Msg,button('Your turn')) ,  
  new(B1,button('')) ,  
  ...  
  new(B9,button('')) ,
```



Cenni a una possibile interfaccia grafica con XPCE

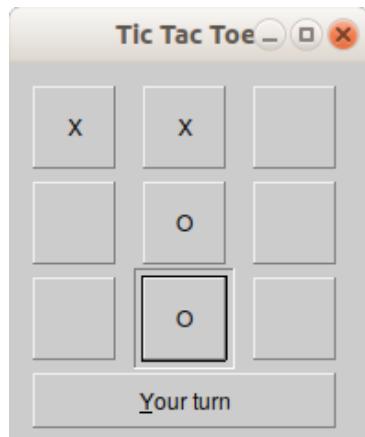


```
gui :-  
  /* costruzione bottoni */  
  
  new(Msg,button('Your_turn')),  
  new(B1,button('')),  
  ...  
  new(B9,button('')),
```

```
/* costruzione finestra */  
new(Dialog, dialog('Tic_Tac_Toe')),  
send(Dialog, gap, size(10,10)),  
send(Dialog, append, B1),  
send(Dialog, append, B2, right),  
send(Dialog, append, B3, right),  
send(Dialog, append, B4, below),  
...  
send(Dialog, append, Msg, below),  
send(Msg, alignment, left),  
send(Dialog, open),  
/* il gioco inizia */  
gui_go(Msg,Buttons).
```



Cenni a una possibile interfaccia grafica con XPCE



```
gui :-  
    /* costruzione bottoni */  
  
    new(Msg,button('Your\turn')),  
    new(B1,button('')),  
    ...  
    new(B9,button('')),
```

```
/* costruzione finestra */  
new(Dialog, dialog('Tic\Tac\Toe')),  
send(Dialog, gap, size(10,10)),  
send(Dialog, append, B1),  
send(Dialog, append, B2, right),  
send(Dialog, append, B3, right),  
send(Dialog, append, B4, below),  
...  
send(Dialog, append, Msg, below),  
send(Msg, alignment, left),  
send(Dialog, open),  
/* il gioco inizia */  
gui_go(Msg,Buttons).
```

```
/* inizializzazione bottoni */  
send(Bi, message,  
    message(@prolog,click,I,B1,B2,...)),  
send(Bi, label, ''),  
send(Bi, size, size(45,45))  
...  
  
/* quando si clicca */  
send(Bi, message,  
    message(@prolog,true)),  
send(Bi, label, 'X'),  
...
```



Compilazione stand-alone della versione grafica

[Paradigma logico](#)

[Prolog](#)

[Costrutti Prolog](#)

[Unificazione](#)

[Conjunctive queries](#)

[Le Regole](#)

[Ragionamento](#)

[Liste](#)

[Applicazioni](#)

[Programmazione
nondeterministica](#)

In che consiste

Tic tac toe

Analisi del gioco

I/O elementare

GUI

[Unicità di Prolog](#)

```
swipl --goal=gui --stand_alone=true --pce -o tic-tac-toe-gui  
      -c tic-tac-toe-gui.pl
```

■ Spiegazione delle opzioni:

- ◆ --pce rende disponibili le primitive per la GUI
- ◆ le altre sono come nell'esempio precedente

■ Attenzione! Se si usa '@' bisogna dichiararlo come operatore unario prefisso (altrimenti viene generato un syntax error)

```
/* inserire all'inizio del programma */  
:- op(1, fx, @).
```



Riassumendo: caratteristiche uniche di Prolog

■ Invertibilità dei predicati

- ◆ un singolo predicato implementa molte funzioni
- ◆ dovuto al fatto che le variabili logiche sono IN/OUT/IN-OUT a seconda dei parametri attuali

[Paradigma logico](#)

[Prolog](#)

[Costrutti Prolog](#)

[Unificazione](#)

[Conjunctive queries](#)

[Le Regole](#)

[Ragionamento](#)

[Liste](#)

[Applicazioni](#)

[Programmazione
nondeterministica](#)

[Unicità di Prolog](#)



Riassumendo: caratteristiche uniche di Prolog

[Paradigma logico](#)

[Prolog](#)

[Costrutti Prolog](#)

[Unificazione](#)

[Conjunctive queries](#)

[Le Regole](#)

[Ragionamento](#)

[Liste](#)

[Applicazioni](#)

[Programmazione
nondeterministica](#)

[Unicità di Prolog](#)

■ Invertibilità dei predicati

- ◆ un singolo predicato implementa molte funzioni
- ◆ dovuto al fatto che le variabili logiche sono IN/OUT/IN-OUT a seconda dei parametri attuali

■ Programmazione nondeterministica

- ◆ con ricerca automatica delle soluzioni
- ◆ basato sul backtracking (cioè il meccanismo di visita dei search tree)



Riassumendo: caratteristiche uniche di Prolog

Paradigma logico

Prolog

Costrutti Prolog

Unificazione

Conjunctive queries

Le Regole

Ragionamento

Liste

Applicazioni

Programmazione
nondeterministica

Unicità di Prolog

■ Invertibilità dei predicati

- ◆ un singolo predicato implementa molte funzioni
- ◆ dovuto al fatto che le variabili logiche sono IN/OUT/IN-OUT a seconda dei parametri attuali

■ Programmazione nondeterministica

- ◆ con ricerca automatica delle soluzioni
- ◆ basato sul backtracking (cioè il meccanismo di visita dei search tree)

■ Restano molti aspetti interessanti che purtroppo non abbiamo il tempo di illustrare

- ◆ strutture dati parziali (permettono append e creazione dizionari in tempo *costante*)
- ◆ meta-predicati / reflection
- ◆ aggregati (setof)
- ◆ forall ...