ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

La Negazione in Prolog: NAF

Federico Chesani

DISI

Department of Informatics – Science and Engineering

Disclaimer & Further Reading

- These slides are largely based on previous work by Prof. Paola Mello

Il Problema della Negazione

- Finora non abbiamo preso in esame il trattamento di informazioni negative

- Abbiamo considerato solo programmi logici, che sono costituiti da clausole definite e che quindi non possono contenere atomi negati. Inoltre, attraverso la risoluzione SLD, non è possibile derivare informazioni negative.

```
persona(maria).
persona(giuseppe).
persona(anna).
cane(fido).
```

Tuttavia, è intuitivo concludere che l'affermazione ***`persona(fido)`*** *sia falsa in quanto non dimostrabile con i fatti contenuti nel programma.*
Sicuri?

Ipotesi di Mondo Chiuso

- Questo approccio è seguito nelle basi di dati in cui, per problemi di dimensioni, si memorizzano solo le informazioni positive.

- Questo significato intuitivo è espresso dalla regola di inferenza nota come **Closed World Assumption** (**CWA**) o Ipotesi di Mondo Chiuso [Reiter '78].
  - Se un atomo "ground" A non è conseguenza logica di un programma P, allora si può inferire ~A

$$\frac{P \not\models A}{\sim A}$$

CWA(P**)** = {~A | A non è conseguenza logica di P }

Ipotesi di Mondo Chiuso

- Esempio:

```
capitale(roma).
citta(X) :- capitale(X).
citta(bologna).
```

- Con la CWA è possibile inferire **`~capitale(bologna).`**

- La CWA è un esempio di regola di inferenza **NON MONOTONA** in quanto l'aggiunta di nuovi assiomi alla teoria (ossia nuove clausole in un programma) può modificare l'insieme di teoremi che valevano precedentemente.

Non-Monotonicità della CWA

- Esempio;

```
capitale(roma).
citta(X):-capitale(X).
citta(bologna).
capitale(bologna). % aggiunto...
```

- Con la CWA non è più possibile inferire **`~capitale(bologna).`**

- L'aggiunta di nuovi assiomi alla teoria (ossia nuove clausole in un programma) ha modificato l'insieme di teoremi che valevano precedentemente.

Ipotesi di Mondo Chiuso

- A causa dell'indecidibilità (**semi-decidibilità**) della logica del primo ordine, **non esiste alcun algoritmo in grado di stabilire in un tempo finito se A non è conseguenza logica di P**
- Dal punto di vista operazionale, se A non è conseguenza logica di P, la risoluzione SLD può non terminare.
- Vediamo un esempio …

CWA – Esempio

```
citta(roma):- citta(roma).
citta(bologna).
```

- `citta(roma)` non è conseguenza logica di P e

  CWA(P) = { `~ citta(roma)` }

  ma la risoluzione SLD per `citta(roma)` non termina.

- L'applicazione della CWA deve necessariamente essere **ristretta agli atomi la cui dimostrazione termina in tempo finito**, cioè agli atomi per cui la risoluzione SLD non diverge.

Negazione per Fallimento

- L'uso della CWA viene sostituito con quello di una regola meno potente, in grado di dedurre un insieme più piccolo di informazioni negative.

- **Negazione per Fallimento ("Negation as Failure" – NF)** [Clark 78], si limita a derivare la negazione di atomi la cui dimostrazione termina con fallimento in tempo finito

- Dato un programma P, se l'insieme FF(P) (insieme di fallimento finito) denota gli atomi A per cui la dimostrazione fallisce in tempo finito, la regola NF si esprime come

$$NF(P)=\{ \sim A \mid A \in FF(P)\}$$

Negazione per Fallimento

- Se un atomo A appartiene a FF(P) allora A non è conseguenza logica di P, ma non è detto che tutti gli atomi che non sono conseguenza logica del programma appartengano all'insieme di fallimento finito.

Esempio:

```
citta(roma) :- citta(roma).
citta(bologna).
```

- **`citta(roma)`** non è conseguenza logica di P, ma non appartiene a FF(P).
- Non riesco a dimostrare comunque **`~citta(roma)`**

Negazione – CWA versus NF

- **Closed World Assumption** (CWA) o Ipotesi di Mondo Chiuso [Reiter '78].
  - Se un atomo "ground" A non è conseguenza logica di un programma P, allora si può inferire ~A

$$\frac{P \not\models A}{\sim A}$$

CWA(P) = {~A | non esiste una refutazione SLD per P∪{A} }

- **Negazione per Fallimento** ("Negation as Failure" - NF) [Clark 78], si limita a derivare la negazione di atomi la cui derivazione termina con fallimento finito

$$NF(P)=\{ \sim A \mid A \in FF(P)\}$$

Esempio – Esame del 11 Settembre 2008

```
no_dupl([], []).
no_dupl([X|Xs], Ys):-
        member(X, Xs),
        no_dupl(Xs, Ys).
no_dupl([X|Xs], [X|Ys]):-
        nonmember(X, Xs),
        no_dupl(Xs, Ys).

nonmember(_, []).
nonmember(X, [Y|Ys]):-
        X \= Y,
        nonmember(X, Ys).
```

Esempio – Esame del 11 Settembre 2008 – Soluzione col not

```
no_dupl([], []).
no_dupl([X|Xs], Ys):-
            member(X, Xs),
            no_dupl(Xs, Ys).
no_dupl([X|Xs], [X|Ys]):-
            not(member(X, Xs)),
            no_dupl(Xs, Ys).
```

Esempio – Esame del 11 Settembre 2008
Soluzione col not

```
no_dupl([], []).
no_dupl([X|Xs], Ys):-
            member(X, Xs),
            no_dupl(Xs, Ys).
no_dupl([X|Xs], [X|Ys]):-
            \+(member(X, Xs)),
            no_dupl(Xs, Ys).
```

- In SICStus prolog, not si scrive \+
- SWISH Prolog supporta entrambe le notazioni

Esempio – Esame del 11 Settembre 2008
Soluzione col cut

```
no_dupl([], []).
no_dupl([X|Xs], Ys):-
            member(X, Xs), !,
            no_dupl(Xs, Ys).
no_dupl([X|Xs], [X|Ys]):-
            no_dupl(Xs, Ys).
```

- Disegnare l'albero SLDNF per goal:

```
:-no_dupl([a,b,b], Y).
```

Risoluzione SLDNF

- Per risolvere goal generali, cioè che possono contenere letterali negativi, si introduce un'estensione della risoluzione SLD, nota come **risoluzione SLDNF** [Clark 78].

- Combina la risoluzione SLD con la negazione per fallimento (NAF, Negation As failure)

Risoluzione SLDNF

- Sia $:- L_1, \ldots, L_m$ il goal (generale) corrente, in cui $L_1, \ldots, L_m$ sono letterali (atomi o negazioni di atomi). Un passo di risoluzione SLDNF si schematizza come segue:

  - Non si seleziona alcun letterale negativo L_i, se non è "ground";
  - Se il letterale selezionato L_i è positivo, si compie un passo ordinario di risoluzione SLD
  - Se L_i è del tipo ~A (con A "**ground**") ed A fallisce finitamente (cioè ha un albero SLD di fallimento finito), L_i ha successo e si ottiene il nuovo risolvente

$$:- L_1, \ldots, L_{i-1}, L_{i+1}, \ldots, L_m$$

Risoluzione SLDNF

- Risolvere con successo un letterale negativo non introduce alcun legame (unificazione) per le variabili dal momento che **si considerano solo letterali negativi "ground"** : al nuovo risolvente

$$:- L_1, \ldots, L_{i-1}, L_{i+1}, \ldots, L_m$$

  non si applica alcuna sostituzione.

- Una regola di calcolo si dice **safe** se seleziona un letterale negativo solo quando è "ground".
- La selezione di letterali negativi solo "ground" è necessaria per garantire correttezza e completezza della risoluzione SLDNF.

Risoluzione SLDNF e Negazione per Fallimento

La risoluzione SLDNF è alla base della realizzazione della negazione per fallimento nei sistemi Prolog

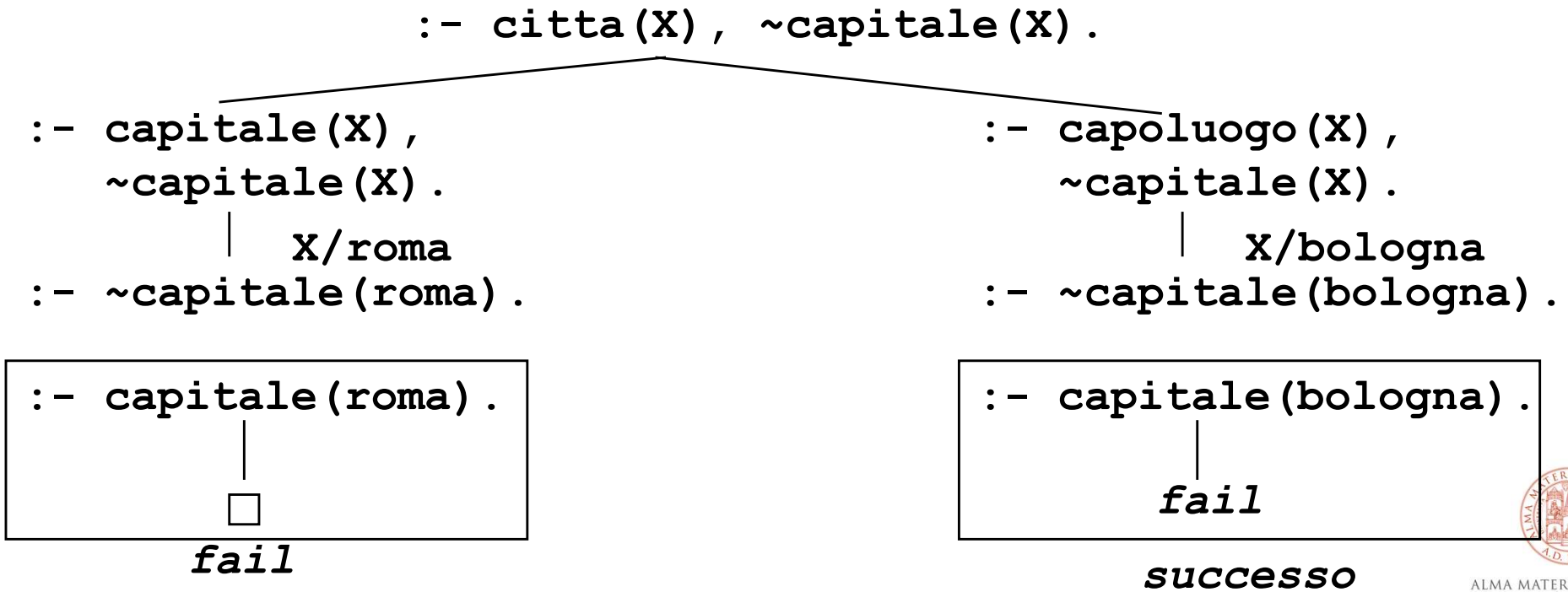
- Per dimostrare ~A, dove A è un atomo, l'interprete del linguaggio cerca di costruire una dimostrazione per A

- Due casi particolari (tre):
  - Se la dimostrazione ha successo, allora la dimostrazione di ~A fallisce;
  - se la dimostrazione per A fallisce finitamente ~A si considera dimostrato con successo.

Risoluzione SLDNF e Negazione per Fallimento

```
capitale(roma).
capoluogo(bologna).
citta(X):- capitale(X).
citta(X):- capoluogo(X).

:- citta(X), ~capitale(X).
```



Risoluzione SLDNF e Negazione per Fallimento

- Il linguaggio Prolog seleziona sempre il letterale più a sinistra, **senza controllare che sia "ground"** e quindi non adotta una regola di selezione safe

- Realizzazione **non corretta** della risoluzione SLDNF

- Cosa succede se si seleziona un letterale negativo non ground ?

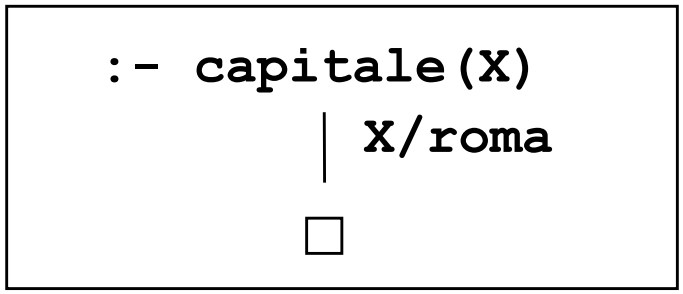
Risoluzione SLDNF e Negazione per Fallimento

```
capitale(roma).
capoluogo(bologna)
citta(X):- capitale(X).
citta(X):- capoluogo(X).

:- ~capitale(X), citta(X).
```

Seleziono il letterale ***~capitale(X)*** *che non è ground al momento della valutazione*

```
:- ~capitale(X), citta(X), .
```



fail

ATTENZIONE: la query esiste un X che non è capitale ma è città risponde NO.
SCORRETTO

Compito del 20 Dicembre 2004

- Si consideri li seguente programma Prolog che calcola se un numero è primo (dove mod calcola il modulo, cioè il resto della divisione intera):

```
primo(N) :- not(divisibile(N)).
divisibile(N) :- compreso(2, M, N), 0 is N mod M.

compreso(I, X, S) :- I>=S, !, fail.
compreso(I, I, S).
compreso(I, X, S) :- J is I+1, compreso(J, X, S).
```

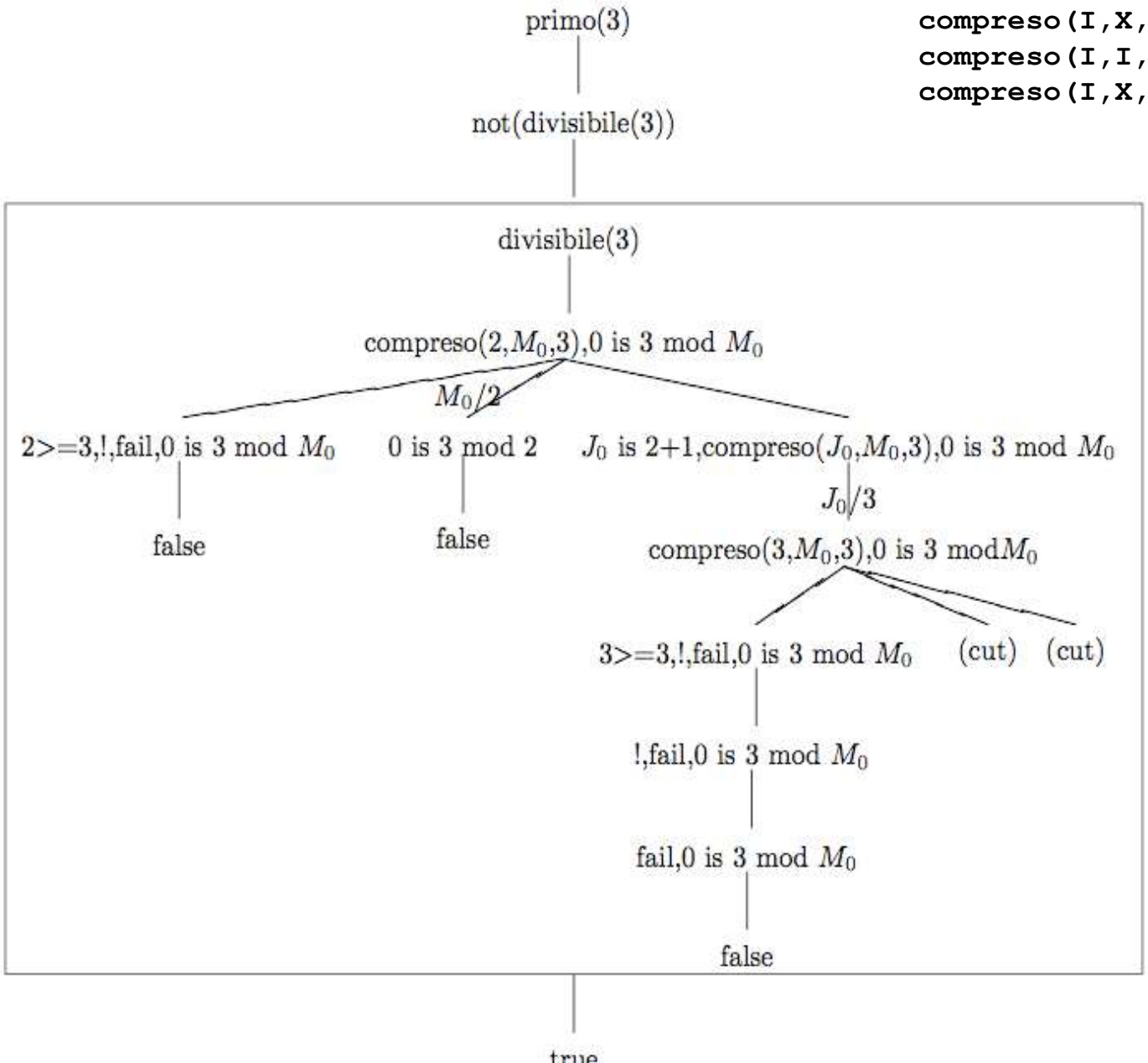
- Si disegni l'albero SLDNF relative al goal: :- primo(3).

Nota: tutti I rami dell'albero SLD nel riquadro devono essere di fallimento affinché il not abbia successo.

Compito del 20 Dicembre 2004 – Soluzione

```
primo(N):- not(divisibile(N)).
divisibile(N):- compreso(2,M,N),0 is N mod M.

compreso(I,X,S):- I>=S, !, fail.
compreso(I,I,S).
compreso(I,X,S):- J is I+1, compreso(J,X,S).
```



Negazione e Quantificatori

- Il problema nasce dal fatto che non si interpreta correttamente la quantificazione nel caso di letterali negativi non "ground"

- Si consideri il programma precedente:

```
capitale(roma).
capoluogo(bologna)
citta(X):- capitale(X).
citta(X):- capoluogo(X).
```

- e il goal G:

```
:-~capitale(X).
```

che corrisponde alla negazione della formula

$$F = \exists \ \mathbf{X\ \sim capitale(X).}$$

Esiste una entità che non è una capitale? (sì, bologna)

Negazione e Quantificatori

- Con la risoluzione SLDNF, si cerca una dimostrazione per

  `:- capitale(X).`

  ossia per

  F' = ∃ **`X capitale(X).`**

- Dopo di che si nega il risultato ottenendo

  F'' = ~ (∃ **`X capitale(X)).`**

  che corrisponde a

  F'' = ∀ **`X (~ capitale(X))`**

- Quindi se esiste una **`X`** che è capitale, F'' fallisce, ma la query originaria era:

  F = ∃ **`X ~capitale(X).`**

Negazione in Prolog

- La forma di negazione introdotta in quasi tutti i linguaggi logici, Prolog compreso, è la negazione per fallimento
- Può essere realizzata facilmente scambiando tra loro la nozione di successo e di fallimento
- Il meccanismo di valutazione di una query negativa ~Q è definito in Prolog nel modo seguente:
  - si valuta la controparte positiva della query Q. Se Q fallisce, si ha successo, mentre se la Q ha successo la negazione fallisce
- Si noti che, data la strategia di risoluzione utilizzata dal Prolog è possibile che la dimostrazione di Q non abbia termine (ossia che il Prolog vada in loop in tale dimostrazione)

Negazione in Prolog

- Prolog adotta una regola di selezione dei letterali left-most (non è safe). Questo può generare problemi perché il significato logico di tali query è diverso da quello atteso

```
:- not p(X).
```

- Il significato di tale query è il seguente

$$\exists \texttt{ X (not p(X))}$$

- Prolog verifica il goal

```
:- p(X).
```

- Il significato di tale query è il seguente

$$\exists \texttt{ X p(X)}$$

Negazione in Prolog

- Dopo di che si nega il risultato, ossia:

  not(∃ X p(X))

- Che corrisponde a:

  ∀ X not(p(X))

- Noi ci chiedevamo, invece:

  ∃ X not(p(X))

Negazione e SWI Prolog e SWIsh – Esempio

- Il Prolog SWI Prolog e SWIsh adotta la convenzione ANSI che indica il not con il simbolo: \+

```
disoccupato(X):- adulto(X),
                 \+(occupato(X)).
occupato(giovanni).
adulto(mario).

?- \+(occupato(giovanni)).
no

?- \+(occupato(mario)).
yes
```

Negazione e SWI Prolog e SWIsh – Esempio

- Consideriamo lo stesso programma:

```
(cl1)        disoccupato(X):- adulto(X),
                              \+(occupato(X)).
(cl2)        occupato(giovanni).
(cl3)        adulto(mario).
```

- E la query (ground):

```
:-disoccupato(mario).
```

Vogliamo sapere se ***mario*** *è* ***disoccupato.***

- E la risposta è **yes**

Negazione e SWI Prolog e SWIsh – Esempio

- Consideriamo lo stesso programma:

```
(cl1)        disoccupato(X):- adulto(X),
                              \+(occupato(X)).
(cl2)        occupato(giovanni).
(cl3)        adulto(mario).
```

  - E la query (non ground):
    `:- disoccupato(X).`
  - Che risposta otteniamo?
  - Yes X=mario

Vogliamo sapere se ***esiste X disoccupato.***

Negazione e SWI Prolog e SWIsh – Esempio

- Consideriamo lo stesso programma:

```
(cl1)        disoccupato(X):- \+(occupato(X)),
                              adulto(X).
(cl2)        occupato(giovanni).
(cl3)        adulto(mario).
```

  - E la query (non ground):
    `:- disoccupato(X).`
  - la risposta è **no**

Vogliamo sapere se ***esiste X disoccupato.***

Negazione e SWI Prolog e SWIsh – Esempio

- Ri-cambiamo ora l'ordine dei letterali nella prima clausola:

```
(cl1)        disoccupato(X):- adulto(X),
                              \+(occupato(X)).
(cl2)        occupato(giovanni).
(cl3)        adulto(mario).
```

  - E la query (non ground):
    `:- disoccupato(X).`

Vogliamo sapere se ***esiste X disoccupato.***

  - Che risposta otteniamo?

  - Dal programma ci aspettiamo e otteniamo come risposta **`yes`** con **`X/mario`**
  - Scambiando i letterali, il letterale negativo viene selezionato quando è ground

Riassumendo

- Prolog non adotta una regola di selezione safe
- Questo fa perdere la correttezza del sistema di dimostrazione
- Usando la regola di selezione di Prolog, si possono ottenere risultati diversi da quelli attesi a causa delle quantificazioni delle variabili.
- E' buona regola di programmazione verificare che i goal negativi siano sempre **ground** al momento della selezione.
- Questo controllo è a carico dell'utente programmatore !! (esistono predicati predefiniti **`var`** e **`nonvar`**)