Esempio compito 11 Sett 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).
```

Compito 11 Sett 2008 – 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).
```

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

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*

IPOTESI DI MONDO CHIUSO

- Questo e' l'approccio 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-MONOTONICITA' DELLA CWA

- Esempio

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

- 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 …

ESEMPIO CWA

```
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.

```
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 - recap

- 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)={ ~A | A ∈ FF(P)}

Esempio compito 11 Sett 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).
```

Compito 11 Sett 2008 – 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).
```

- *Albero SLDNF per goal:*

```
:-no_dupl([1,2,1,2,1], 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 $:-L_1,...,L_m$ il goal (generale) corrente, in cui $L_1,...,L_m$ 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

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

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

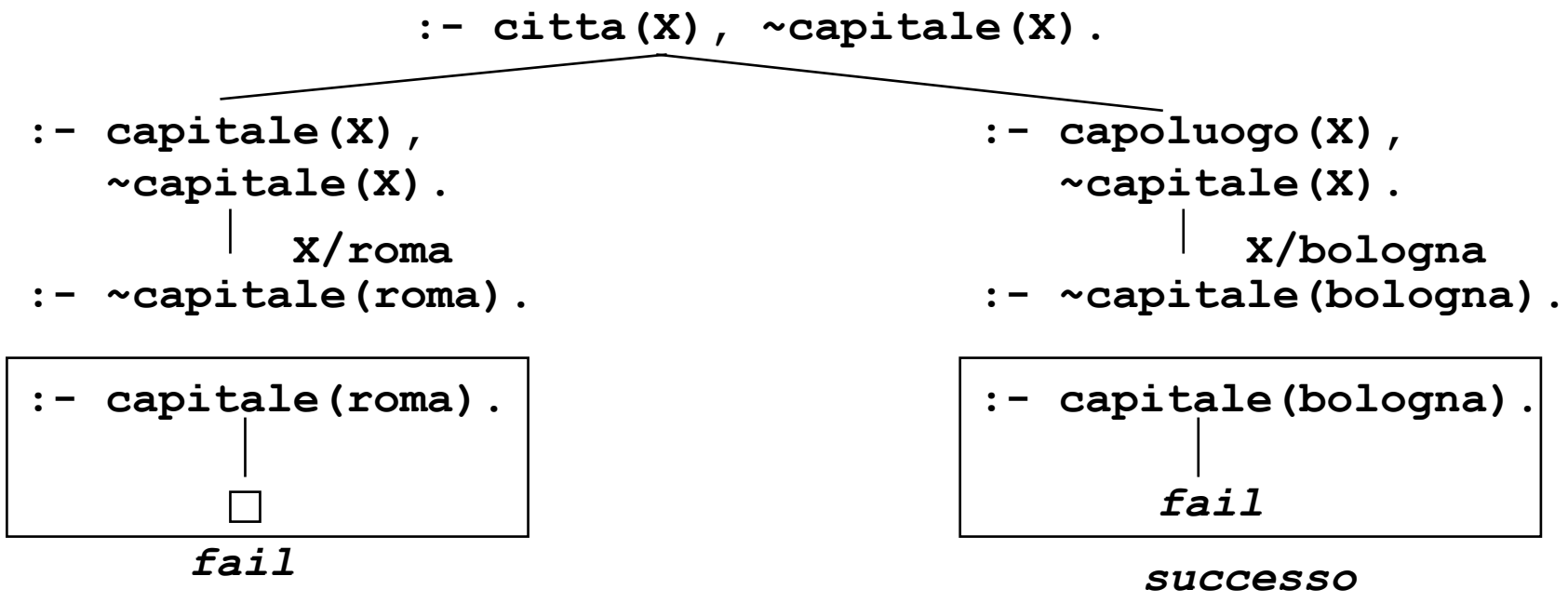
RISOLUZIONE SLDNF E NEGAZIONE PER FALLIMENTO

- 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
- Se la dimostrazione ha successo, allora la dimostrazione di ~A fallisce, mentre 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).
```

Seziono il letterale ***~capitale(X)*** *che non e' ground al momento della valutazione*

```
:- ~capitale(X), citta(X), .
        |
   :- capitale(X)
          | X/roma
          □
```

fail

ATTENZIONE*: la query esiste un X che non e' capitale ma e' citta' risponde NO.*
SCORRETTO

NEGAZIONE E QUANTIFICATORI

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

- Si consideri il programma precedente e il goal G:

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

che corrisponde alla negazione della formula

F = **∃ X ~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

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

- Prolog verifica il goal

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

- Il significato di tale query è il seguente

$$\exists X \ p(X)$$

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 SICStus Prolog

- SICStus Prolog installato in Laboratorio adotta la convenzione ANSI che indica il not con il simbolo: \+

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

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

Negazione e SICStus Prolog

- SICStus Prolog installato in Laboratorio adotta la convenzione ANSI che indica il not con il simbolo: \+

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

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

ESEMPIO

- Consideriamo lo stesso programma:

```
(cl1) disoccupato(X):- adulto(X),
                       \+(occupato(X)).

(cl2) occupato(giovanni).
(cl3) adulto(mario).
```

Vogliamo sapere se ***mario*** *e'* ***disoccupato.***

- E la query (ground):

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

- E la risposta è **yes**

ESEMPIO

- Consideriamo il seguente programma:

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

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

Vogliamo sapere se esiste un ***X*** *tale che* ***disoccupato(X)*** *.*

- Che risposta otteniamo?

ESEMPIO

- Consideriamo il seguente programma:

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

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

Vogliamo sapere se esiste un ***X*** *tale che* ***disoccupato(X)*** *.*

- Che risposta otteniamo?

- Yes X=mario

ESEMPIO

- Consideriamo lo stesso programma, ma con i due sottogoal nel corpo della clausola scambiati:

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

Vogliamo sapere se ***esiste X che*** *e'* ***disoccupato.***

- e la query (ground):

```
:-disoccupato(X).
```

- la risposta è **`no`**

ESEMPIO

- 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: `:- disoccupato(X).`

Vogliamo sapere se esiste un ***X*** *tale che* ***disoccupato(X)*** *.*

- Dal programma ci aspettiamo e otteniamo come risposta **`yes`** con **`X/mario`**

Scambiando i letterali, il letterale negativo viene selezionato quando e' ground

ESEMPIO

- Consideriamo il seguente programma:

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

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

Vogliamo sapere se esiste un ***X*** *tale che* ***disoccupato(X)*** *.*

- Che risposta otteniamo?

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`**)