

IL PROBLEMA DELLA NEGAZIONE

- Finora non abbiamo preso in esame il trattamento di informazioni negative
- Solo programmi logici 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) .`

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

$$\text{CWA}(P) = \{ \sim A \mid \text{non esiste una refutazione SLD per } P \cup \{A\} \}$$

IPOTESI DI MONDO CHIUSO

- Esempio
capitale (roma) .
citta (X) :- capitale (X) .
citta (bologna) .
- Con la CWA e' possibile inferire \sim capitale (bologna) .
- La CWA e' 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.

IPOTESI DI MONDO CHIUSO

- A causa dell'indecidibilità 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
- Infatti, dal punto di vista operativo, se A non è conseguenza logica di P la risoluzione SLD può non terminare
- L'applicazione della CWA deve necessariamente essere ristretta agli atomi la cui prova termina in tempo finito, cioè agli atomi per cui la risoluzione SLD non diverge.

IPOTESI DI MONDO CHIUSO

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

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

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

$$\text{CWA}(P) = \{ \sim \text{citta}(\text{roma}) \}$$

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

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 le negazioni 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

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

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 e' conseguenza logica di P ma non appartiene a $FF(P)$.

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 (NF)

RISOLUZIONE SLDNF

- Sia $:-\mathbf{L}_1, \dots, \mathbf{L}_m$ il goal (generale) corrente, in cui $\mathbf{L}_1, \dots, \mathbf{L}_m$ sono letterali (atomi o negazioni di atomi). Un passo di risoluzione SLDNF si schematizza come segue:
 - Non si seleziona alcun letterale negativo \mathbf{L}_i , se non è "ground";
 - Se il letterale selezionato \mathbf{L}_i è positivo, si compie un passo ordinario di risoluzione SLD
 - Se \mathbf{L}_i è del tipo $\sim A$ (con A "ground") ed A fallisce finitamente (cioè ha un albero SLD di fallimento finito), \mathbf{L}_i ha successo e si ottiene il nuovo risolvete

$$:- \mathbf{L}_1, \dots, \mathbf{L}_{i-1}, \mathbf{L}_{i+1}, \dots, \mathbf{L}_m$$

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 risolvete

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

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 $\sim A$, 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 $\sim A$ fallisce, mentre se la dimostrazione per A fallisce finitamente $\sim A$ 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) .
```

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

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

```
:- capitale(roma) .
```

□

fail

```
:- capoluogo(X),  
   ~capitale(X) .  
   | X/bologna  
:- ~capitale(bologna) .
```

```
:- capitale(bologna) .
```

fail

successo

RISOLUZIONE SLDNF E NEGAZIONE PER FALLIMENTO

- Il linguaggio Prolog non adotta una regola di selezione *safe*, cioè seleziona sempre il letterale più a sinistra, senza controllare che sia "ground"
- 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 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 negati non "ground"
- Si consideri il programma precedente e il goal G:
`:- ~capitale (X) .`

che corrisponde alla negazione della formula

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

Esiste una entità (bologna) che non è una capitale.

NEGAZIONE E QUANTIFICATORI

- Con la risoluzione SLDNF, si cerca una dimostrazione per

$$:- \text{capitale}(X) .$$

ossia per

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

- Dopo di che si nega il risultato ottenendo

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

che corrisponde a

$$F = \forall X (\sim \text{capitale}(X))$$

- Quindi se esiste una x che e' capitale, F fallisce

NEGAZIONE IN PROLOG

- La forma di negazione introdotta in quasi tutti i linguaggi logici, Prolog compreso, e' 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 $\sim Q$ 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 strategia di selezione dei letterali left-most. 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:

$$\text{not } (\exists \mathbf{x} \text{ p } (\mathbf{x}))$$

- Che corrisponde a:

$$\forall \mathbf{x} \text{ not } (\text{p } (\mathbf{x}))$$

- Noi ci aspettavamo:

$$\exists \mathbf{x} \text{ not } (\text{p } (\mathbf{x}))$$

ESEMPIO

- Consideriamo il seguente programma:

```
(c11) disoccupato(X) :- not occupato(X),  
                        adulto(X) .
```

```
(c12) occupato(giovanni) .
```

```
(c13) adulto(mario) .
```

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

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

- Dal programma ci aspettiamo come risposta **yes** con **x/mario**
- Invece otteniamo come risposta **no**

ESEMPIO

- Consideriamo il seguente programma:

```
(c11) disoccupato(X) :- not occupato(X),  
                        adulto(X) .
```

```
(c12) occupato(giovanni) .
```

```
(c13) adulto(mario) .
```

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

*Vogliamo sapere se
mario e'
disoccupato.*

- E la risposta e' `yes`

ESEMPIO

- Cambiamo ora l'ordine dei letterali nella prima clausola:

```
(c11) disoccupato (X) :- adulto (X) ,  
                             not occupato (X) .
```

```
(c12) occupato (giovanni) .
```

```
(c13) 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*

RIASSUMENDO

- Prolog non adotta una regola di selezione SAFE
- Usando la regola di selezione del 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 e' a carico dell'utente !!