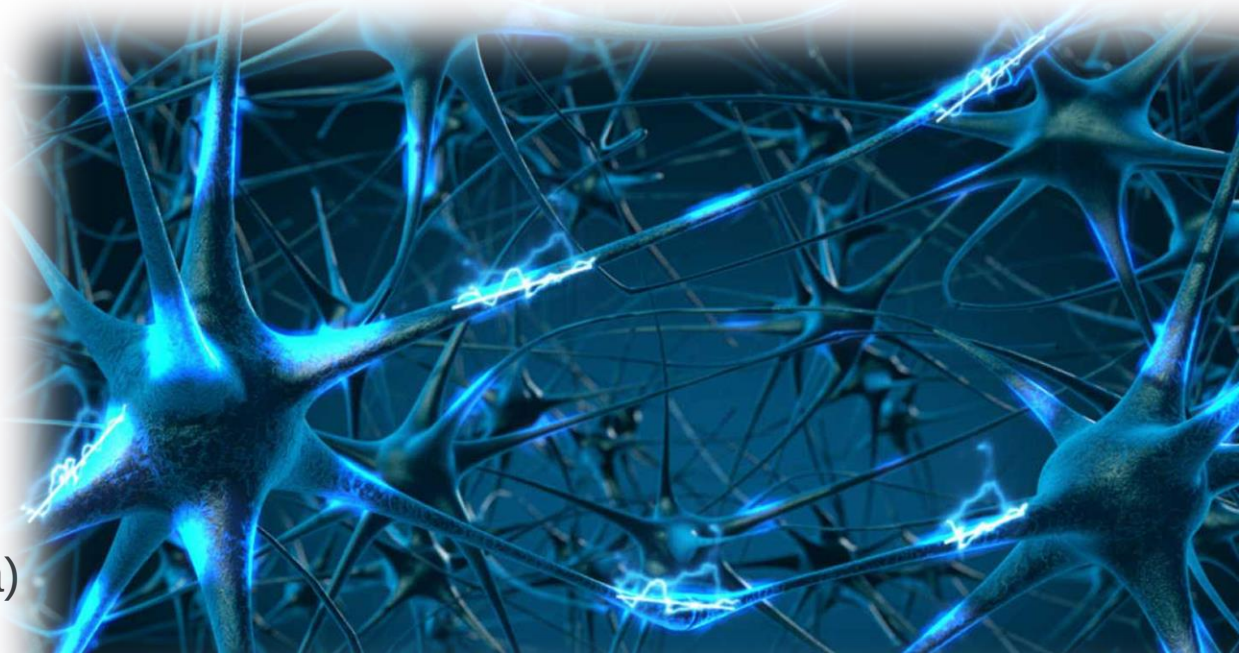
INTELLIGENZA ARTIFICIALE

KR: INFERENZA NEL CALCOLO DEI PREDICATI (*)

Corsi di Laurea in Informatica, Ing. Gestionale, Ing. Informatica, Ing. di Internet

(a.a. 2022-2023)

Roberto Basili



(*) alcune *slides* sono di
Maria Simi (Univ. Pisa)

Overview

- Introduzione alle strategie di risoluzione efficenti per il FOL
- Le clausole di Horn
- La risoluzione SLD per la programmazione logica
- Elementi di programmazione logica
- Fatti e Regole come KB di un agente
- Elementi extra-logici del Prolog

Nel libro … AIMA 3° edition

9 INFERENCE IN FIRST-ORDER LOGIC

In which we define effective procedures for answering questions posed in first-order logic.

Chapter 7 showed how sound and complete inference can be achieved for propositional logic. In this chapter, we extend those results to obtain algorithms that can answer any answerable question stated in first-order logic. Section 9.1 introduces inference rules for quantifiers and shows how to reduce first-order inference to propositional inference, albeit at potentially great expense. Section 9.2 describes the idea of **unification**, showing how it can be used to construct inference rules that work directly with first-order sentences. We then discuss three major families of first-order inference algorithms. **Forward chaining** and its applications to **deductive databases** and **production systems** are covered in Section 9.3; **backward chaining** and **logic programming** systems are developed in Section 9.4. Forward and backward chaining can be very efficient, but are applicable only to knowledge bases that can be expressed as sets of Horn clauses. General first-order sentences require resolution-based **theorem proving**, which is described in Section 9.5.

Risoluzione efficiente

- Il *metodo di risoluzione* per il FOL
  - KB in forma a clausole
  - Unificazione e regola di risoluzione
- Sorgenti di non determinismo
  - Scelta delle clausole risolventi
  - Unificazione efficiente
- Come si può rendere più efficiente?
  - Strategie di risoluzione: tecniche per esplorare in maniera efficiente il grafo di risoluzione, possibilmente senza perdere completezza

Strategie di risoluzione

- Si distingue tra (Genesereth&Nilsson, 1987):
  - Strategie di cancellazione
  - Strategie di restrizione
  - Strategie di ordinamento

Strategie di cancellazione

- Si tratta di rimuovere dalla KB (ai fini della dimostrazione) certe clausole che non potranno essere utili nel processo di risoluzione
  1. Clausole con *letterali puri*
  2. *Tautologie*
  3. Clausole *sussunte*

Cancellazione di letterali puri

- Clausole con *letterali puri*: quelli che non hanno il loro negato nella KB

  Es. {¬P, ¬Q, R} {¬P, S} {¬Q, S} {P} {Q} {¬R}

- Tali clausole non potranno mai essere risolte con altre clausole per ottenere { } a causa dei loro letterali puri

Cancellazione di tautologie

- *Tautologie*: clausole che contengono due letterali identici e complementari

  Es. {P(A), ¬P(A), …} {P(x), Q(y), ¬Q(y)}

  La loro rimozione non influenza la soddisfacibilità.

- *Nota*: non basta che siano unificabili e di segno opposto

  Es. {¬P(A), P(x)} {P(A)} {¬P(B)} è insoddisfacibile
  {P(A)} {¬P(B)} non lo è

- Le tautologie possono essere generate ⇒ controllo da fare ad ogni passo

Cancellazione di clausole sussunte

3. Eliminazione di *clausole sussunte* (implicate)
   - Es. P(x) *sussume* P(A), P(A) *sussume* P(A) $\vee$ P(B)
   - In generale: α *sussume* β sse $\exists\sigma\ \alpha\sigma \subseteq \beta$

     se un'istanza di α (con la sost. σ) è un sottoinsieme di β

   Es. {P(x), Q(y)} *sussume* {P(A), Q(v), R(w)} infatti

   {P(x), Q(y)}{x/A, y/v}={P(A), Q(v)}

- β può essere ricavata da α. Quindi β può essere eliminata senza perdere soluzioni.
- Le clausole sussunte possono essere generate.

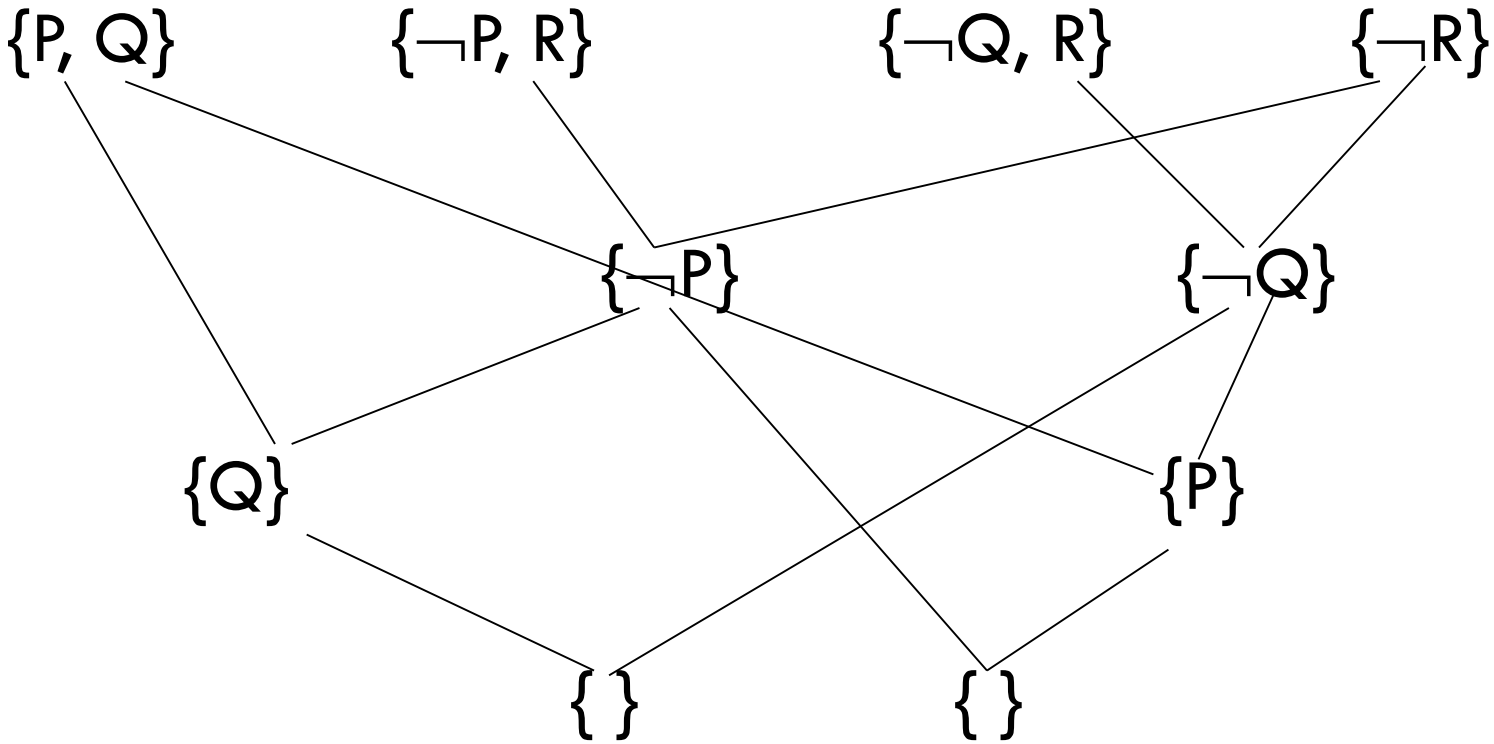
Strategie di restrizione

- Ad ogni passo si sceglie tra un sottoinsieme delle possibili clausole

- Tra le strategie di restrizione possibili:
  1. Risoluzione unitaria
  2. Risoluzione da input
  3. Risoluzione lineare
  4. Risoluzione lineare da input
  5. Risoluzione guidata dal goal

Risoluzione unitaria

- Risoluzione unitaria: almeno una delle due clausole è *unitaria* (contiene un solo letterale)



Risoluzione unitaria: completa?

- Facile da implementare, si converge rapidamente
- *Problema*: la strategia non è completa

  Esempio. $\{P, Q\}\ \{\neg P, Q\}\ \{P, \neg Q\}\ \{\neg P, \neg Q\} \vdash_{RES} \{\ \}$ ma non con risoluzione unitaria
- La strategia è completa per clausole Horn.

  Clausole Horn: clausole con al più un letterale positivo
- Nota: {P, Q} non è una clausola Horn

Risoluzione da input

- Una delle clausole appartiene alla KB iniziale
  Teorema: c'è una risoluzione da input sse ce n'è una unitaria (metodi diversi ma equivalenti)

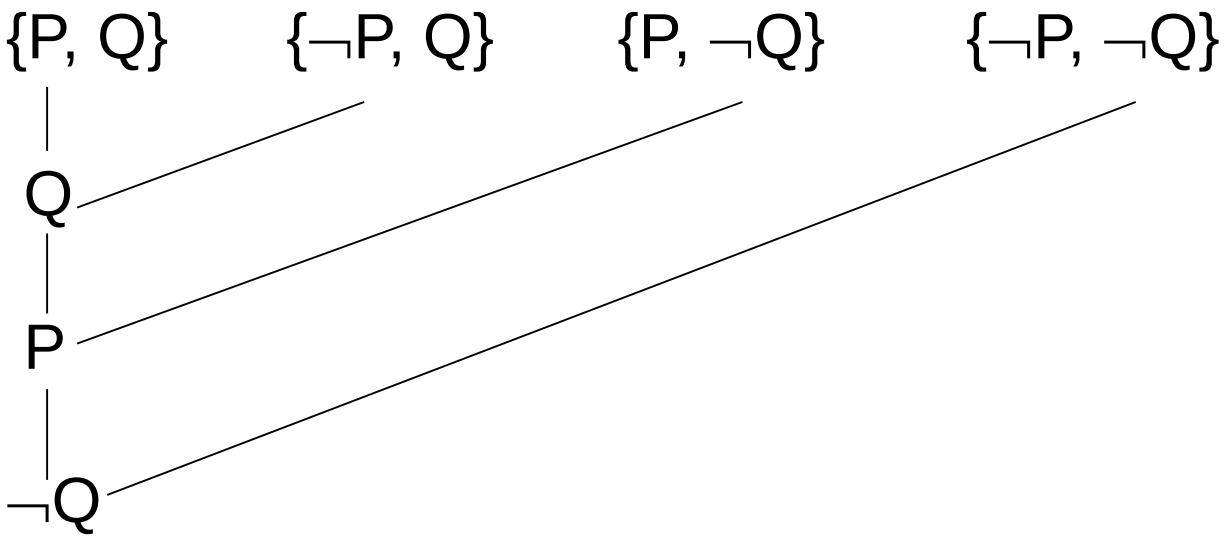
- *Corollario*: risoluzione da input non completa, ma completa per clausole Horn.

  Es. {P, Q} {¬P, Q} {P, ¬Q} {¬P, ¬Q} non Horn

  … e la clausola vuota non può essere generata da input.

Risoluzione lineare da input

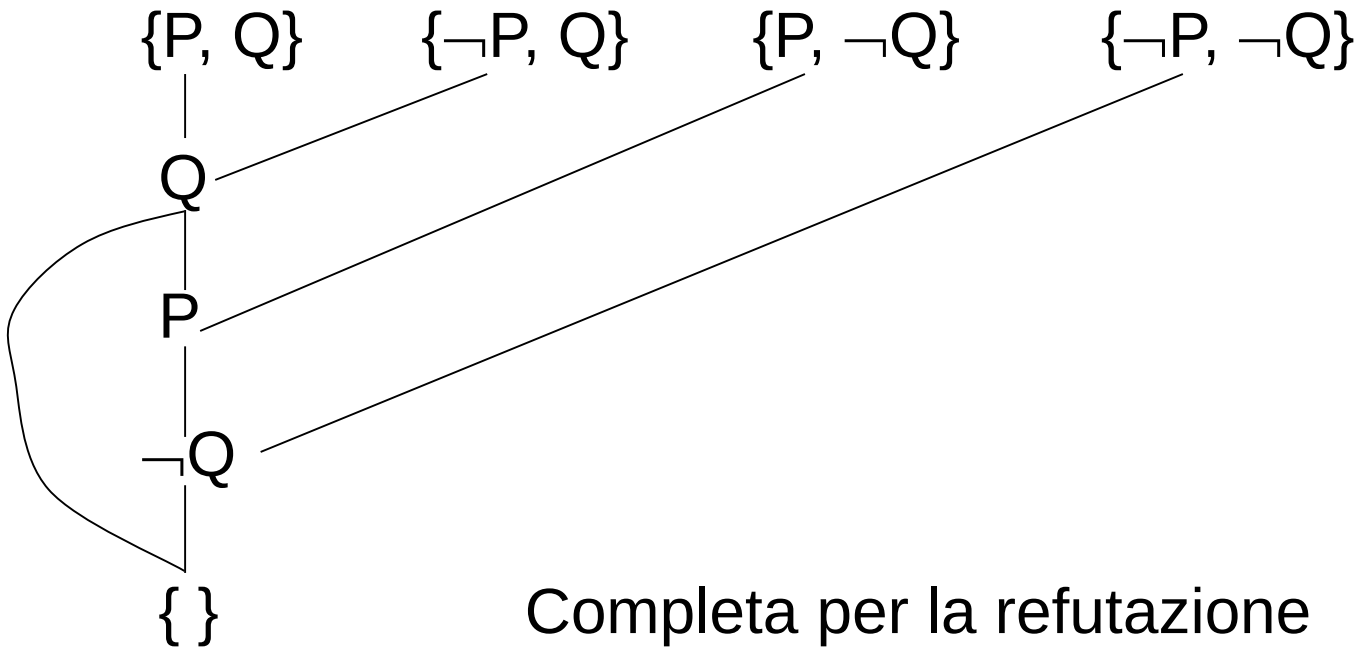
- Una clausola da input con l'ultima clausola generata
- Generalizzazione della risoluzione da input, con in più il vincolo di linearità



- Completa, ma solo per clausole Horn

Risoluzione lineare

- Ultima clausola generata con una clausola da input oppure una clausola antenata.

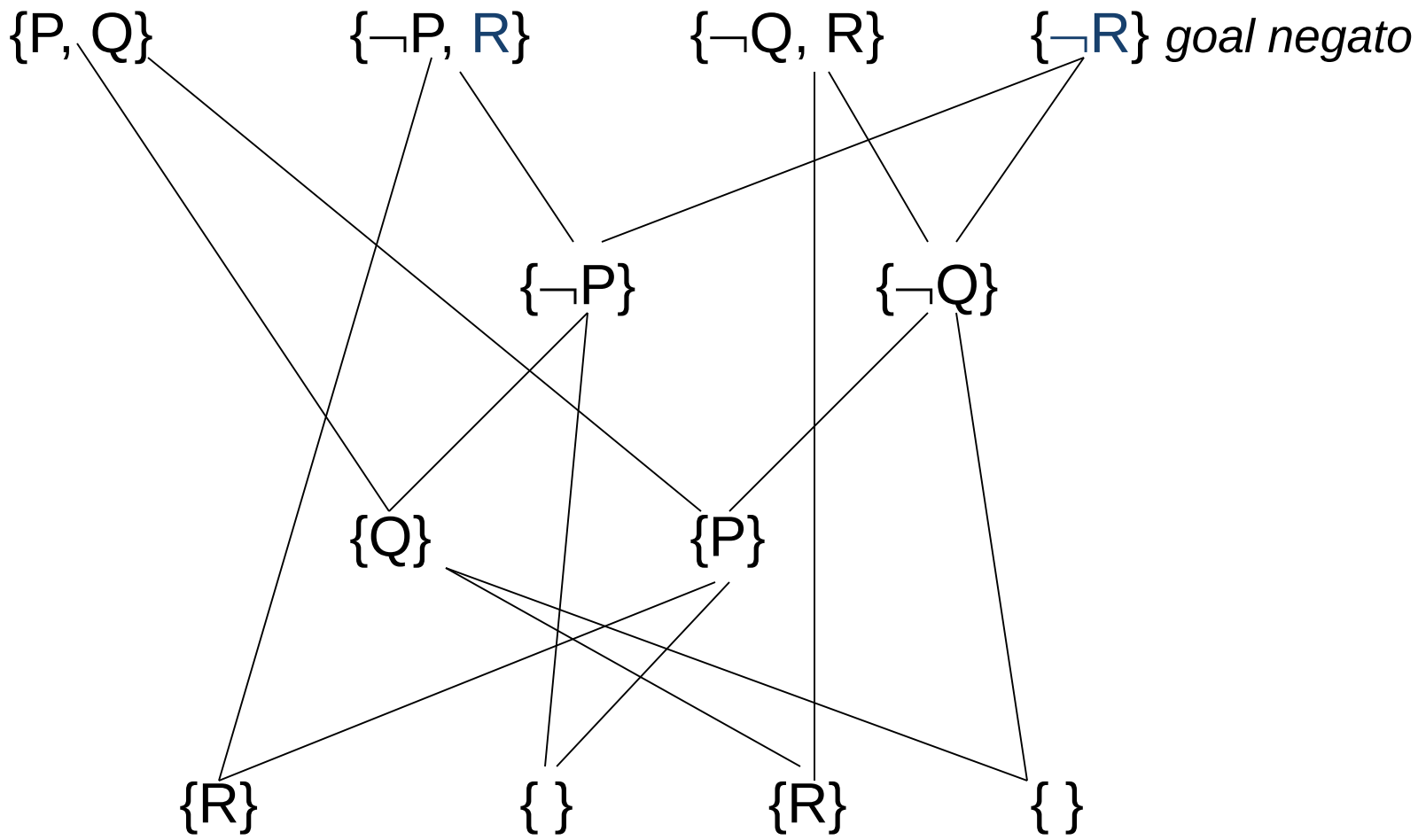


Completa per la refutazione

Risoluzione guidata dal goal

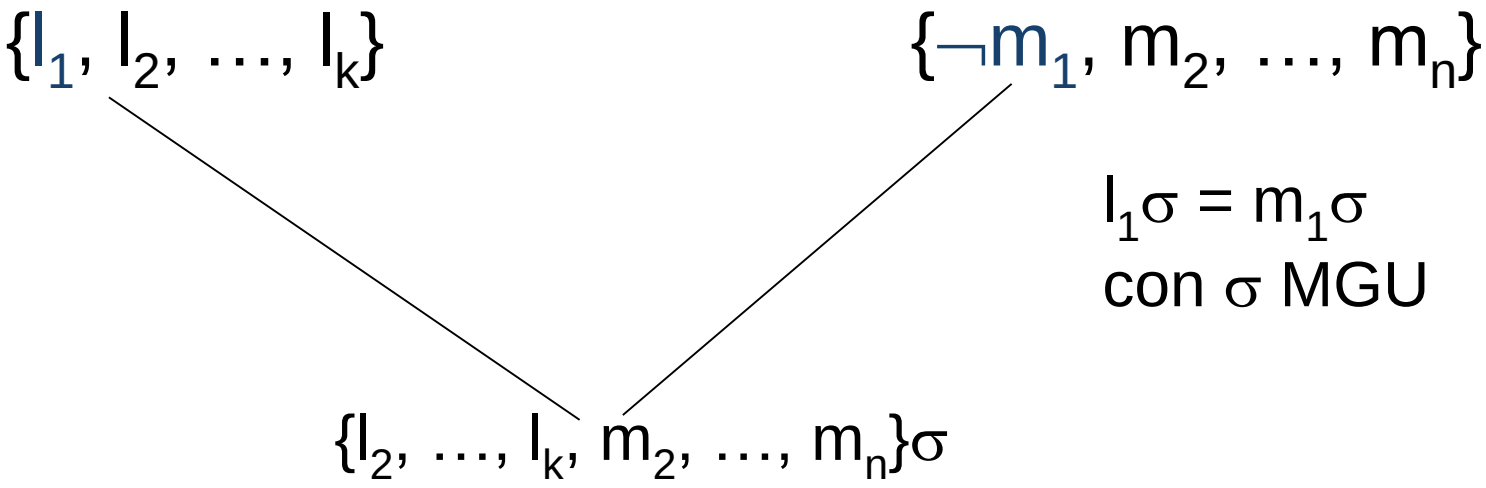
- Insieme di supporto: un sotto-insieme della KB responsabile dell'insoddisfacibilità
- Almeno una delle due clausole appartiene a questo insieme o a suoi discendenti
- Tipicamente, assumendo la KB iniziale consistente, si sceglie come insieme di supporto iniziale il negato della clausola goal
- … è come procedere all'indietro dal goal

Risoluzione all'indietro dal goal: esempio

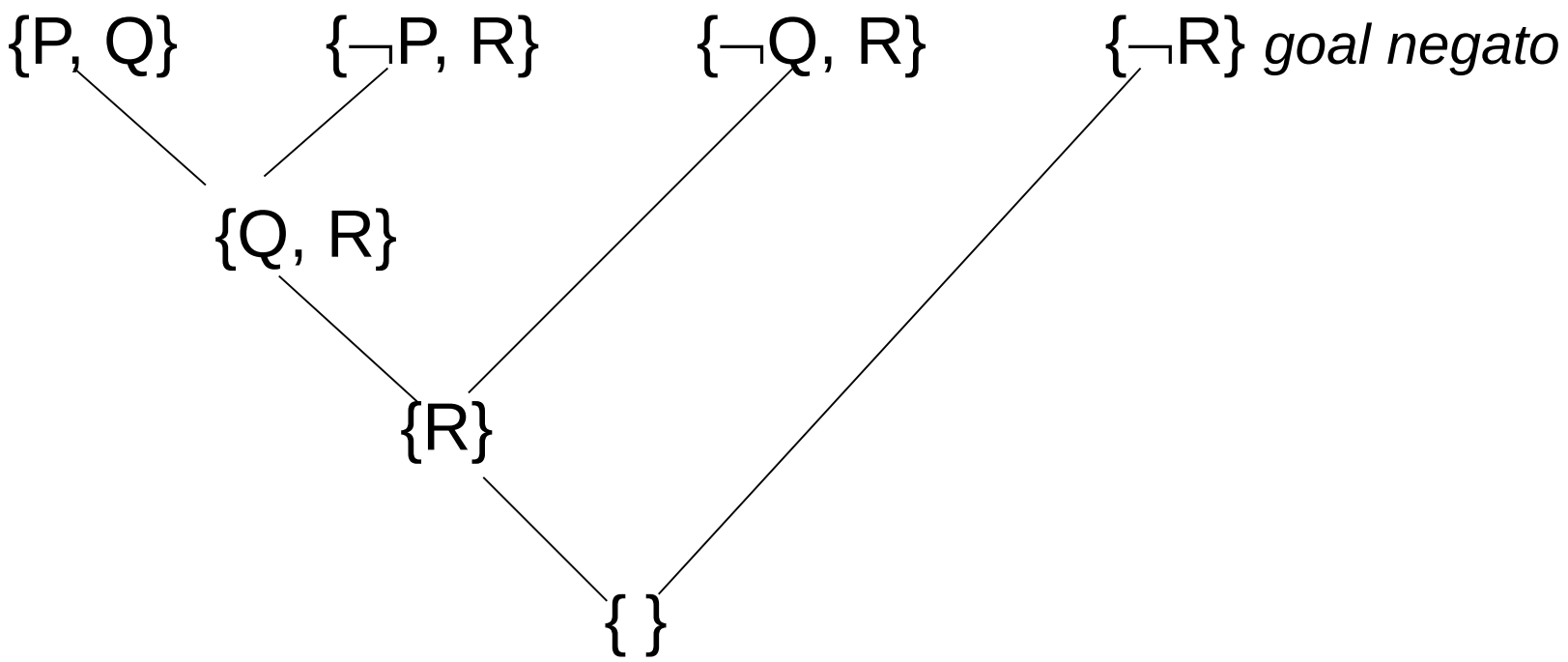


Risoluzione ordinata

- Ogni clausola è un insieme ordinato di letterali e si possono unificare solo i letterali di testa delle clausole
- L'ordinamento deve essere rispettato nel risolvente



Risoluzione ordinata: esempio



La risoluzione ordinata è completa per clausole Horn

Il sottoinsieme “a regole” del FOL

- *Clausole Horn definite: esattamente un letterale positivo*
- Possono essere riscritte come fatti e regole:

$\neg P_1 \vee \ldots \vee \neg P_k \vee Q$

$\neg(P_1 \wedge \ldots \wedge P_k) \vee Q$

$P_1 \wedge \ldots \wedge P_k \Rightarrow Q$ *regola*

Q *fatto*

Sistemi a regole logici

- KB a regole
  - Fatti: letterali positivi. Es. p
  - Regole: $p_1 \wedge p_2 \wedge \ldots \wedge p_n \Rightarrow q$

- Se la KB contiene solo clausole Horn *definite* i meccanismi inferenziali sono molto più semplici, il processo molto più “guidato” senza rinunciare alla completezza.

- Nota: è restrittivo. Non coincide con FOL.

Uso delle regole in avanti e all'indietro

- Concatenazione all'indietro (*Backward Chaining*): un'istanza di ragionamento guidato dall'obiettivo
  - Le regole sono applicate alla rovescia
  - Programmazione logica (PROLOG)
- Concatenazione in avanti (*Forward Chaining*): un'istanza di ragionamento|ricerca guidato dai dati
  - Le regole sono applicate nel senso "antecedente-conseguente"
  - Basi di dati deduttive e sistemi di produzione

Programmazione logica

- I programmi logici sono KB costituiti di clausole Horn definite espressi come fatti e regole, con una sintassi alternativa

  $\{A\}$

  $\{A, \neg B_1, \neg B_2, \ldots, \neg B_n\}$ $\quad$ $[B_1 \wedge B_2 \wedge \ldots \wedge B_n \Rightarrow A]$

  diventano

  A. $\quad$ fatto

  $A :- B_1, B_2, \ldots, B_n$. $\quad$ regola, con *testa* A, il conseguente

- *Altre convenzioni*: in PL le variabili sono indicate con lettere maiuscole, le costanti con lettere minuscole

Programmi logici

- *Interpretazione dichiarativa* di una regola

  $A :- B_1, B_2, \ldots, B_n$ (*A testa*, $B_1, B_2, \ldots B_n$ *corpo*)

  A è vero se sono veri $B_1, B_2, \ldots B_n$
- *Interpretazione procedurale*: la testa può essere vista come una chiamata di procedura e il corpo come una serie di procedure da eseguire in sequenza
- *Clausole goal*

Se $B_1 \wedge B_2 \wedge \ldots \wedge B_n$	è il goal
$\neg(B_1 \wedge B_2 \wedge \ldots B_n) \vee False$	è il goal negato, ovvero
$B_1 \wedge B_2 \wedge \ldots B_n \Rightarrow False$	che viene scritto
$:- B_1, B_2, \ldots, B_n$	omettendo il conseguente

Esempio di KB come programma logico

1. *Genitore(X, Y) :– Padre(X, Y).*
2. *Genitore(X, Y) :– Madre(X, Y).*
3. *Antenato(X, Y) :– Genitore(X, Y).*
4. *Antenato(X, Y) :– Genitore(X, Z), Antenato(Z, Y).*
5. *Padre(gio, mark).*
6. *Padre(gio, luc).*
7. *Madre(lia, gio).*

8. *:– Antenato(lia, mark)* *goal negato*

Risoluzione SLD

- La risoluzione SLD (Selection Linear Definite-clauses) è una strategia *ordinata*, basata su un *insieme di supporto* (la clausola goal), *lineare da input*.
- La risoluzione SLD è completa per clausole Horn.

Alberi di risoluzione SLD

- Dato un programma logico P, l'albero SLD per un goal G è definito come segue:
  - ogni nodo dell'albero corrisponde a un goal [congiuntivo]
  - la radice è *:-G*, il nostro goal
  - sia $:- G_1, G_2, \ldots, G_k$ un nodo dell'albero; il nodo ha tanti discendenti quanti sono i fatti e le regole in *P* la cui testa è unificabile con G_1

    Se $A :- B_1, \ldots, B_k$ e A è unificabile con G_1 il discendente è il goal $:- (B_1, \ldots, B_k, G_2, \ldots, G_k)\gamma$ con $\gamma = MGU(A, G_1)$

  - i nodi che sono clausole vuote sono successi
  - i nodi che non hanno successori sono fallimenti

Esempio di albero SLD: il programma

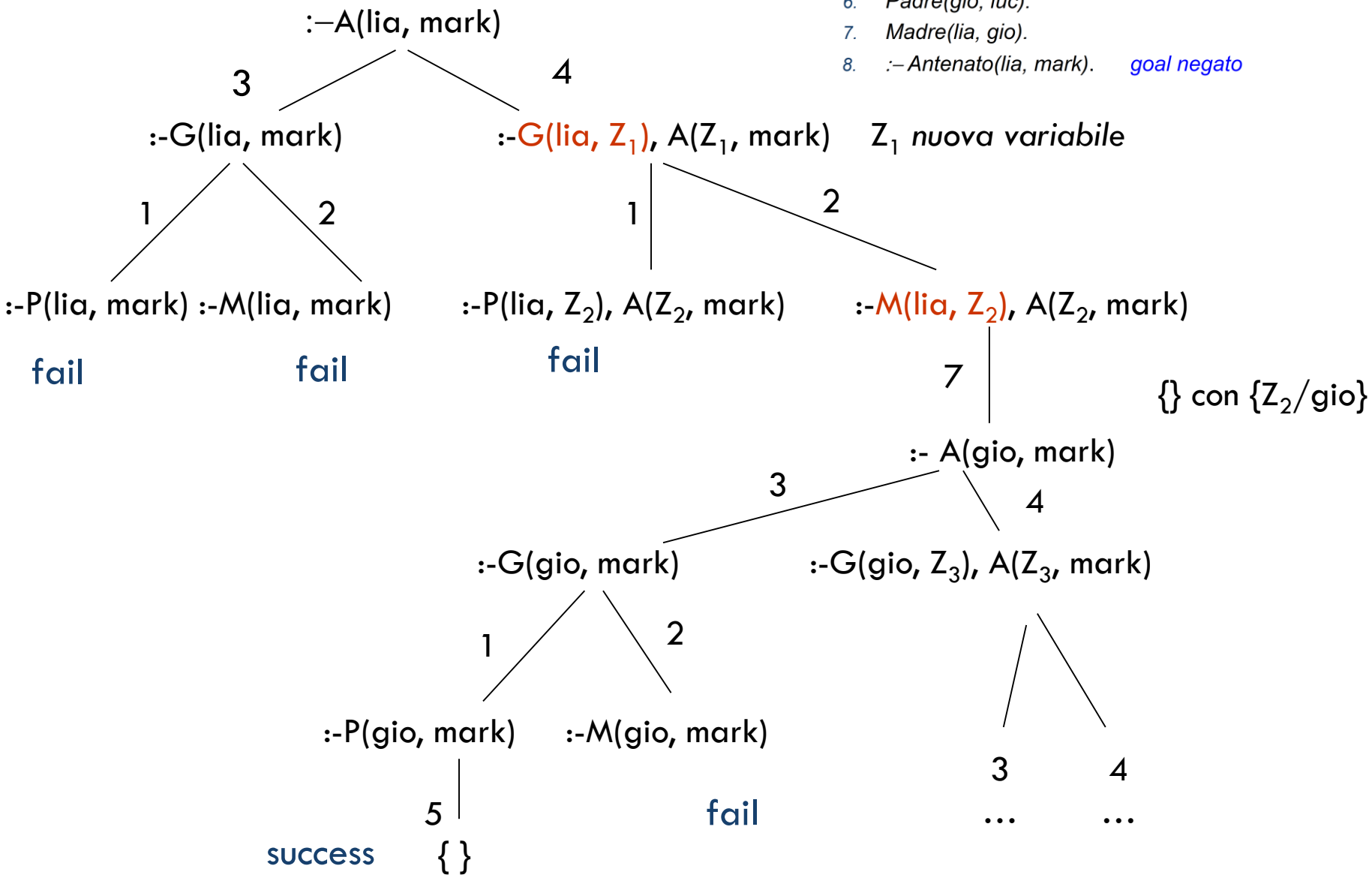
1. *Genitore(X, Y) :– Padre(X, Y).*
2. *Genitore(X, Y) :– Madre(X, Y).*
3. *Antenato(X, Y) :– Genitore(X, Y).*
4. *Antenato(X, Y) :– Genitore(X, Z), Antenato(Z, Y).*
5. *Padre(gio, mark).*
6. *Padre(gio, luc).*
7. *Madre(lia, gio).*

8. *:– Antenato(lia, mark)*. *goal negato*

Albero SLD per:

:- Antenato(lia, mark)

1. *Genitore(X, Y) :– Padre(X, Y).*
2. *Genitore(X, Y) :– Madre(X, Y).*
3. *Antenato(X, Y) :– Genitore(X, Y).*
4. *Antenato(X, Y) :– Genitore(X, Z), Antenato(Z, Y).*
5. *Padre(gio, mark).*
6. *Padre(gio, luc).*
7. *Madre(lia, gio).*
8. *:– Antenato(lia, mark).* *goal negato*



Risoluzione SLD

- La strategia è completa per clausole Horn definite e quindi, se $P \cup \{\neg G\}$ è insoddisfacibile, allora una delle foglie deve essere la clausola vuota (successo)

- Non è restrittivo andare in ordine nel risolvere i sottogoal in congiunzione logica (and).

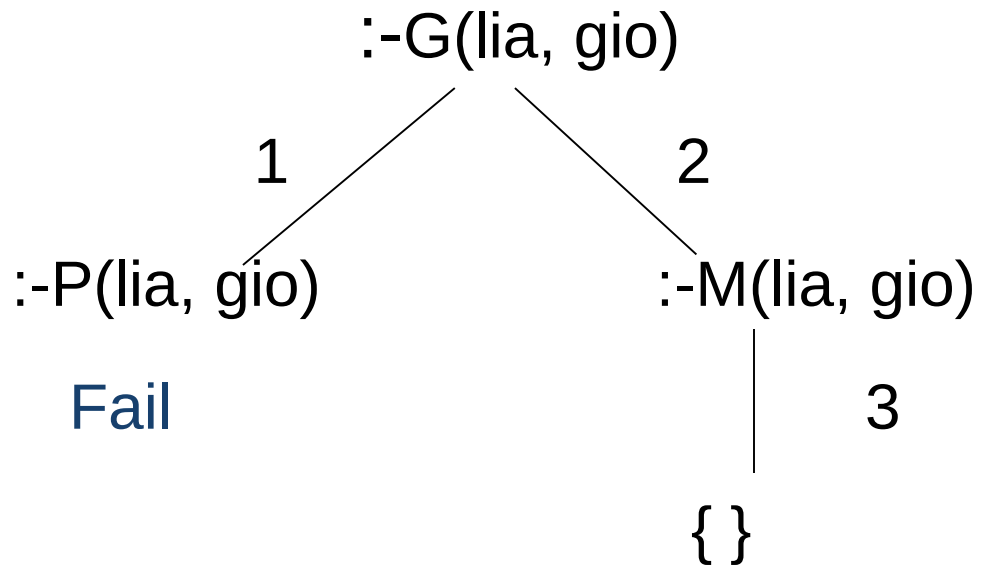
- La sostituzione corrispondente è la *risposta calcolata*

Strategia di visita dell'albero SLD e PROLOG

- A seconda di come visito l'albero potrei anche non trovare la clausola vuota. La strategia di ricerca può essere responsabile dell'incompletezza.
- In PROLOG, il più famoso linguaggio di programmazione logica, la visita dell'albero di risoluzione avviene con una ricerca in profondità, con *backtracking* in caso di fallimento
- Su richiesta si trovano tutte le soluzioni.
- Quindi la strategia di PROLOG non è completa
- PROLOG omette l'*occur check* per motivi di efficienza
- Le regole vengono applicate nell'ordine in cui sono immesse

PROLOG e domande del tipo “si-no”

I numeri corrispondono all’ordine di visita



:- G(lia, gio) → SI

:- G(lia, pete) → NO

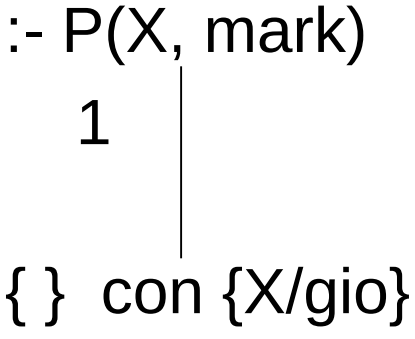
Assunzione di mondo chiuso

PROLOG con domande del tipo “trova”

:- P(X, mark)

chi è il padre di Mark?

X=gio

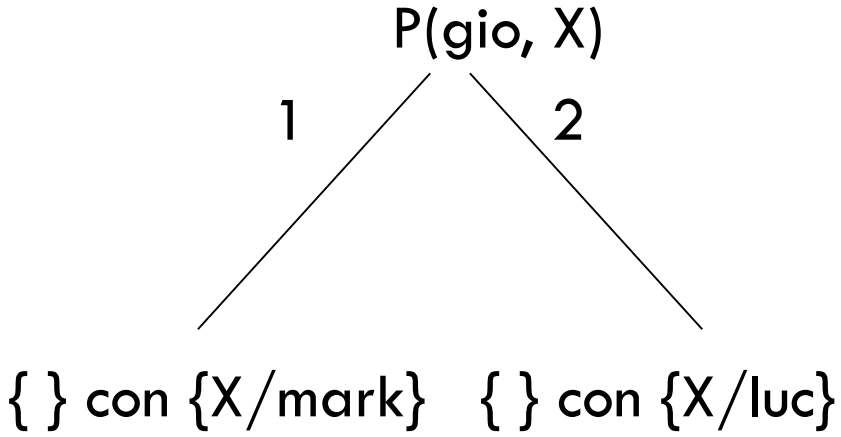


:- P(gio, X)

chi sono i figli di Gio?

X=mark;

X=luc.



Altre domande ...

- Chi è figlio di chi?

  :- G(X, Y).

- Quali sono i fratelli (coloro che hanno lo stesso genitore)?

  :- G(X, Y), G(X, Z).

- Chi sono i nipoti di Lia (in quanto nonna)?

  :- G(lia, X), G(X, Y).

Incompletezza

Supponiamo di avere un programma leggermente diverso:

1. G(X, Y) :- P(X, Y)
2. G(X, Y) :- M(X, Y)
4. A(X, Y) :- A(Z, Y), G(X, Z)
3. A(X, Y) :- G(X, Y)
5. P(gio, mark)
6. P(gio, luc)
7. M(lia, gio)

Nota. Abbiamo scambiato la regola 3 con la 4 e i due letterali nel corpo della 4 tra di loro

Goal:

:- A(lia, mark)

:- A(Z_1, mark), G(lia, Z_1)

:- A(Z_2, mark), G(Z_1, Z_2)

:- A(Z_3, mark), G(Z_2, Z_3)

...

Si finisce in un cammino infinito e non si trova mai la soluzione

Estensioni: le liste

- Prolog ammette anche le liste come strutture dati.
  - [E|L] indica una lista il cui primo elemento è E e il resto è L; [ ] lista vuota.

- Concatenazione di liste A e B in una lista C, i.e. concatena(+A, +B, -C). Il codice Prolog è:

concatena ([ ], Y, Y).

concatena([A|X], Y, [A|Z]) :- concatena(X, Y, Z).

Negazione come fallimento finito

- *Orfano(X) :- not Padre(Y, X)*
- Se *:- Padre(Y, X)* fallisce (non si trovano padri), la risposta è SI
- Non coincide con la negazione logica:
  - $KB \not\vdash Padre(joe, mark)$ piuttosto che
    $KB \vdash \neg Padre(joe, mark)$
- È una forma di ragionamento non monotòno e fa uso della assunzione di mondo chiuso.

Estensioni: semplice aritmetica

- Operatori infissi predefiniti: +, -, *, /, //, ** ...
- Espressioni numeriche: il predicato “A is 2*3” è vero se A ha un valore e il valore di A è 6
- Operatori di confronto: >, <, >=, =<, =:=, =\= forzano la valutazione, variabili ok purché instanziate

  Nota: 2+1 = 1+2 unificazione fallisce; 2+1 =:= 1+2 ok
- Esempio:

  max(X, Y, Y) :- X =< Y.

  max(X, Y, X) :- X>Y.

  Molto elegante, ma presuppone che i primi due argomenti nel goal, X e Y, siano numeri

Per provare …

- SWI Prolog
  http://www.swi-prolog.org/

SummarAIzing

- Abbiamo visto come la risoluzione sia un meccanismo efficiente per la gestione della conseguenza logica in FOL
- La complessità dei processi di risoluzione che sono altamente non deterministici può essere governata ricorrendo ad alcune stategie principali:
  - La cancellazione di clausole che non hanno impatto sui modelli
  - La restrizione a clausole più utili alla inferenza
  - L'ordinamento delle clausole
  - La gestione anticipata di clausole più semplici
- Tra le strategie di restrizione possibili ci sono:
  - La *risoluzione unitaria* dà priorità alle clausole caratterizzate da un solo letterale
  - Nella *risoluzione da input* il risolvente viene ricercato nella clausole della base di KB in input
  - Nella *risoluzione lineare da input* l'ultimo goal generato viene usato per la ricerca dei risolventi nella base di KB in input
  - Infine nella *risoluzione guidata dal goal* il goal obbiettivo della dimostrazione (o confutazione) è usato come primo risolvente
- L'utilizzo di sole clausole di Horn in una base di conoscenza rende la risoluzione completa, grazie alla monotonicità del numero dei letterali attivi per una risoluzione (che decresce sempre)

SummarAIzing (2)

- La risoluzione SLD (Selection Linear Definite-clauses) è il meccanismo di risoluzione efficiente impiegato per la programmazione logica che è quindi ristretto a programmi fatti di clausole di Horn definite
- La programmazione logica definisce la KB in base a fatti (clausole senza letterali negati, cioè implicanti) e regole (clausole con uno o più implicanti)
- Questi costituiscono la base per la rappresentazione del dominio/ambiente, delle regole di evolzione di questo e della pianificazione KB di un agente
- Il Prolog è un linguaggio molto diffuso per la programmazione logica e grazie ai mccanismi di unificazione e sostituzione ha come effetto collaterale la riscrittura dei simli (ad es. variabili in termini) che costituisce un calcolo
- Gli elementi extra-logici del Prolog consentono di controllare la efficienza della sua risoluzione anche se modificano la semantica basata sui modelli che non corrisponde più a quella dei programmi logici. Esempi sono le interazioni con la KB, i meta-predicati, i predicati di azione sulle strutture (ad es. liste) ed infine i controlli (ad esempio il cut/0).