

# Agenti logici: sistemi a regole

*Maria Simi*  
*a.a. 2021-2022*

# Risoluzione efficiente

- Il *metodo di risoluzione* per il FOL
  - KB in forma a clausole
  - Unificazione e regola di risoluzione (estensione rispetto alla regola per PROP)
- Come si può rendere più efficiente?
  - Strategie di risoluzione: tecniche per esplorare in maniera efficiente il grafo di risoluzione, possibilmente senza perdere completezza
  - Un percorso che ci porterà a giustificare i sistemi a regole e le restrizioni del FOL associate.

# Strategie di risoluzione

- Si distingue tra:
  - Strategie di cancellazione: *ci sono clausole che posso eliminare?*
  - Strategie di restrizione: *posso usare ad ogni passo solo alcune clausole?*
  - Strategie di ordinamento: *posso 'risolvere' i letterali in un ordine specifico?*
- Il tutto possibilmente senza perdere completezza.

NOTA: questa parte non la trovate su AIMA ma su [Genesereth-Nilsson]

# Strategie di cancellazione

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

# Cancellazione di clausole: letterali puri

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

Es.  $\{\neg P, \neg Q, R\}$   $\{\neg P, S\}$   $\{\neg Q, S\}$   $\{P\}$   $\{Q\}$   $\{\neg R\}$

- Nell'esempio la seconda e terza clausola.
- Le clausole con letterali puri non potranno mai essere risolte con altre clausole per ottenere  $\{\}$ , tanto vale eliminarle
- Non si perdono soluzioni

# Cancellazione di tautologie

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

Es.  $\{P(A), \neg P(A), \dots\}$     $\{P(x), Q(y), \neg Q(y)\}$

La loro rimozione non influenza la soddisfacibilità.

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

$\{\neg P(A), P(x)\}$     $\{P(A)\}$     $\{\neg P(B)\}$  è insoddisfacibile

$\{P(A)\}$     $\{\neg P(B)\}$  è soddisfacibile

- Le tautologie possono essere generate  $\Rightarrow$  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*  $\{P(A), P(B)\}$

- In generale:  $\alpha$  *sussume*  $\beta$  sse  $\exists \sigma \alpha\sigma \subseteq \beta$

se un'istanza di  $\alpha$  (con la sost.  $\sigma$ ) è un sottoinsieme di  $\beta$

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

$$\{P(x), Q(y)\}\{x/A, y/v\} = \{P(A), Q(v)\} \subseteq \{P(A), Q(v), R(w)\}$$

- Se  $\alpha$  *sussume*  $\beta$ ,  $\beta$  può essere ricavata da  $\alpha$ . Quindi  $\beta$  può essere eliminata senza perdere soluzioni.
- Le clausole sussunte possono essere generate.

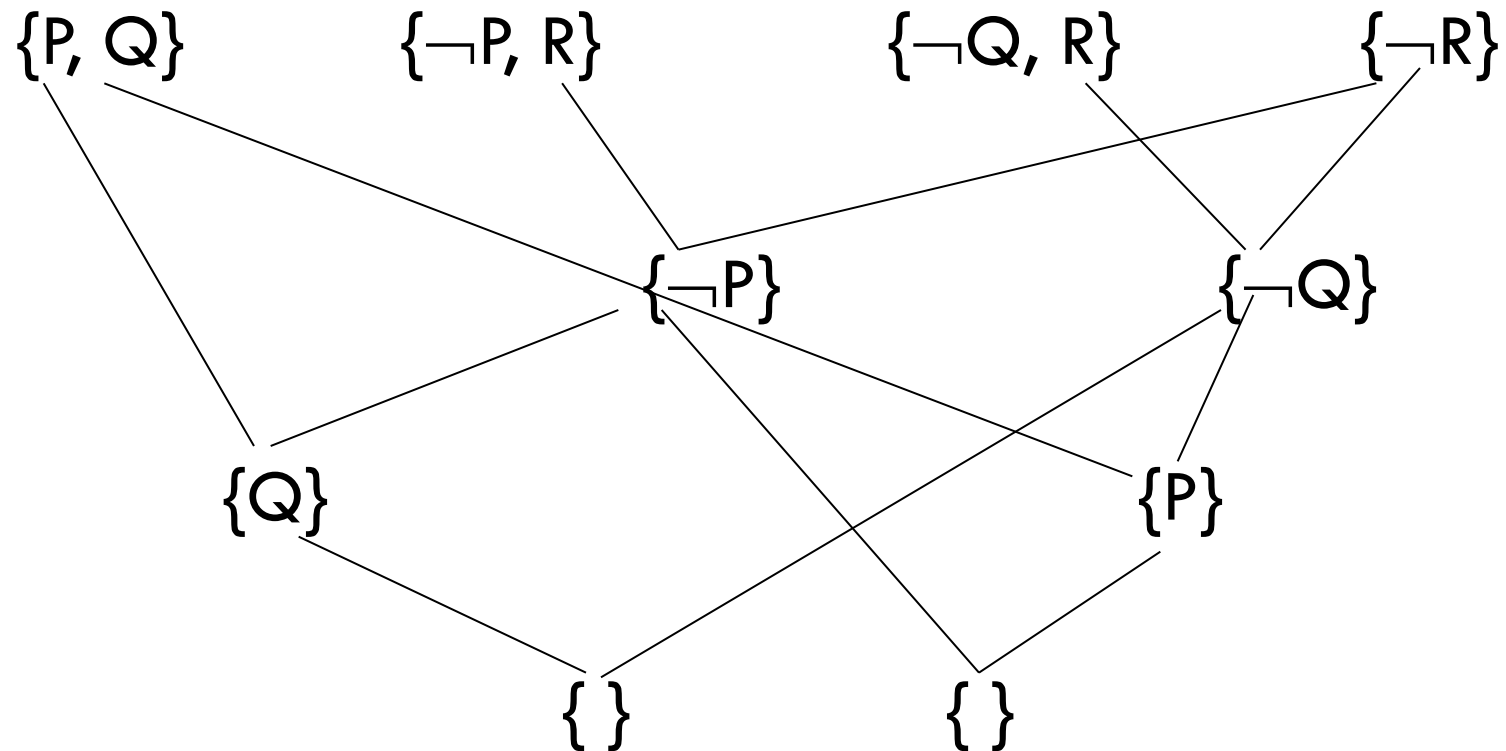
# Strategie di restrizione

- Ad ogni passo si sceglie tra un sottoinsieme delle possibili clausole
- Tra le strategie di restrizione possibili:
  1. Risoluzione unitaria
  2. Risoluzione lineare
  3. 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.  $\{P, Q\} \{\neg P, Q\} \{P, \neg Q\} \{\neg P, \neg Q\} \vdash_{\text{RES}} \{ \}$

ma non con risoluzione unitaria

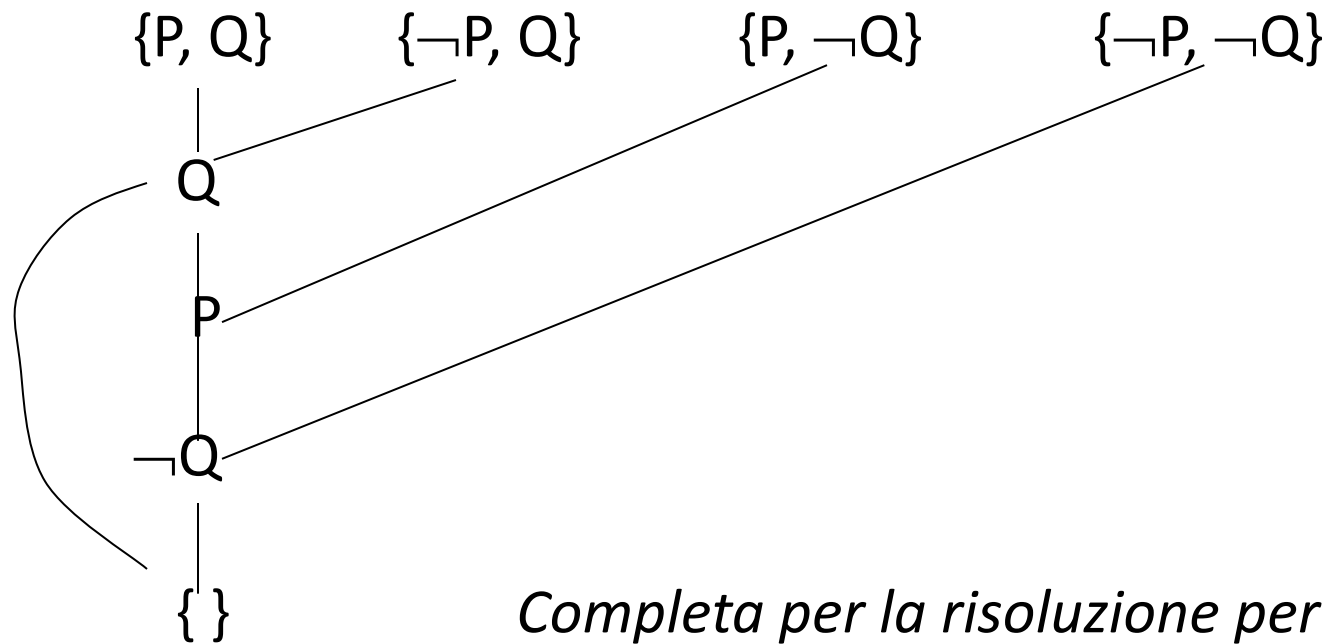
- La strategia è completa per clausole Horn.

*Clausole Horn*: clausole con **al più** un letterale positivo

- Nota:  $\{P, Q\}$  non è una clausola Horn

# Risoluzione lineare

- Ultima clausola generata con una clausola da input (dalla KB iniziale) oppure una clausola antenata.

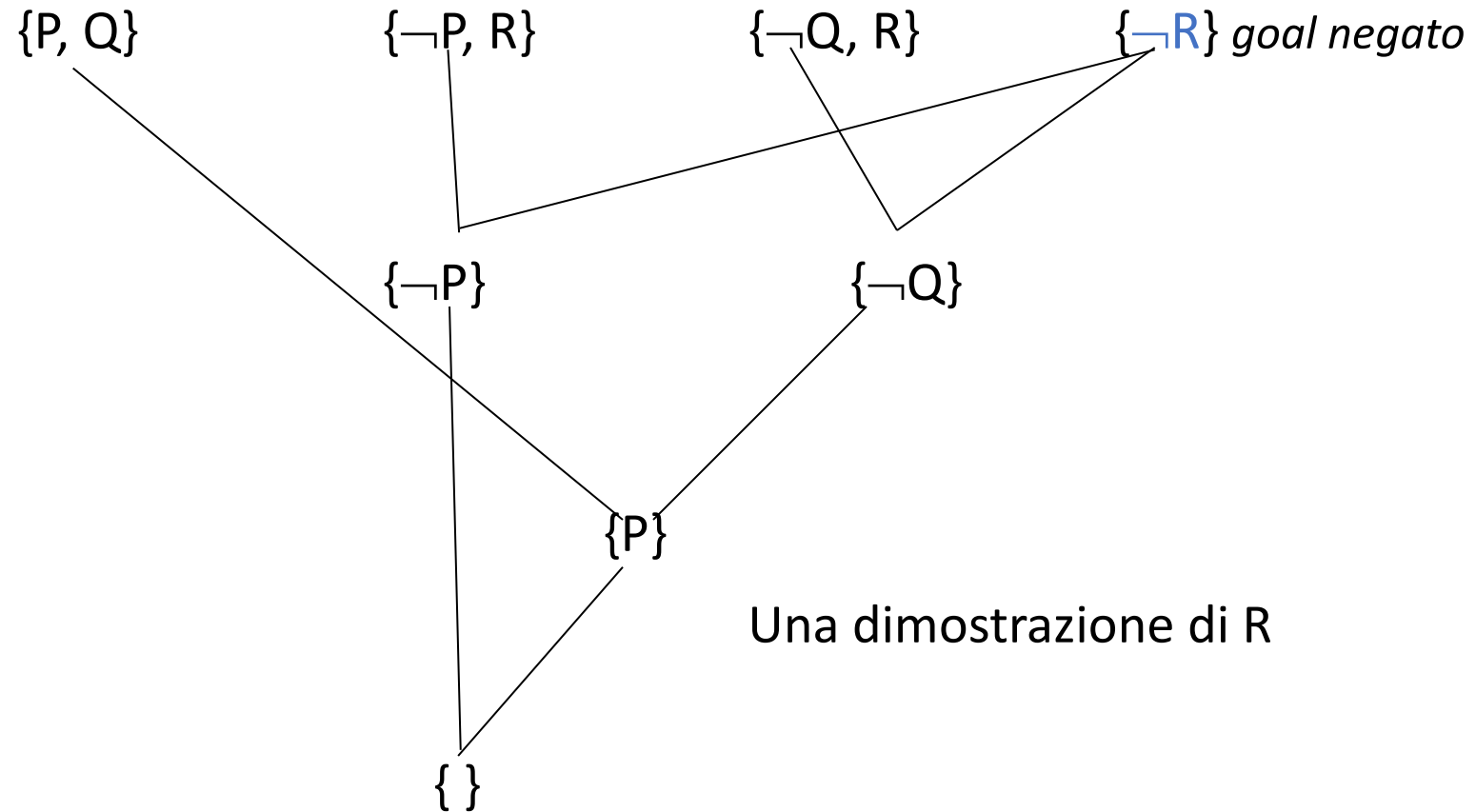


*Completa per la risoluzione per refutazione*

# Risoluzione guidata dal goal

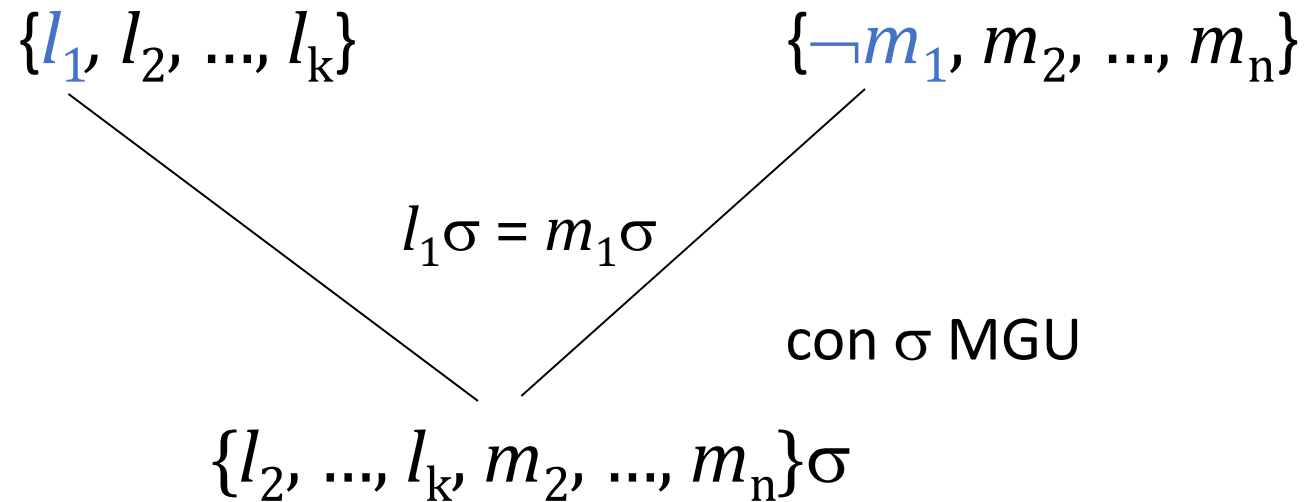
- **Insieme di supporto:** un sotto-insieme della KB responsabile dell'insoddisfacibilità
- Almeno una delle due clausole appartiene a questo insieme o a suoi discendenti
- Tipicamente, assumendo la KB iniziale consistente, si sceglie come insieme di supporto iniziale **il negato della clausola goal**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

# Risoluzione all'indietro dal goal: esempio

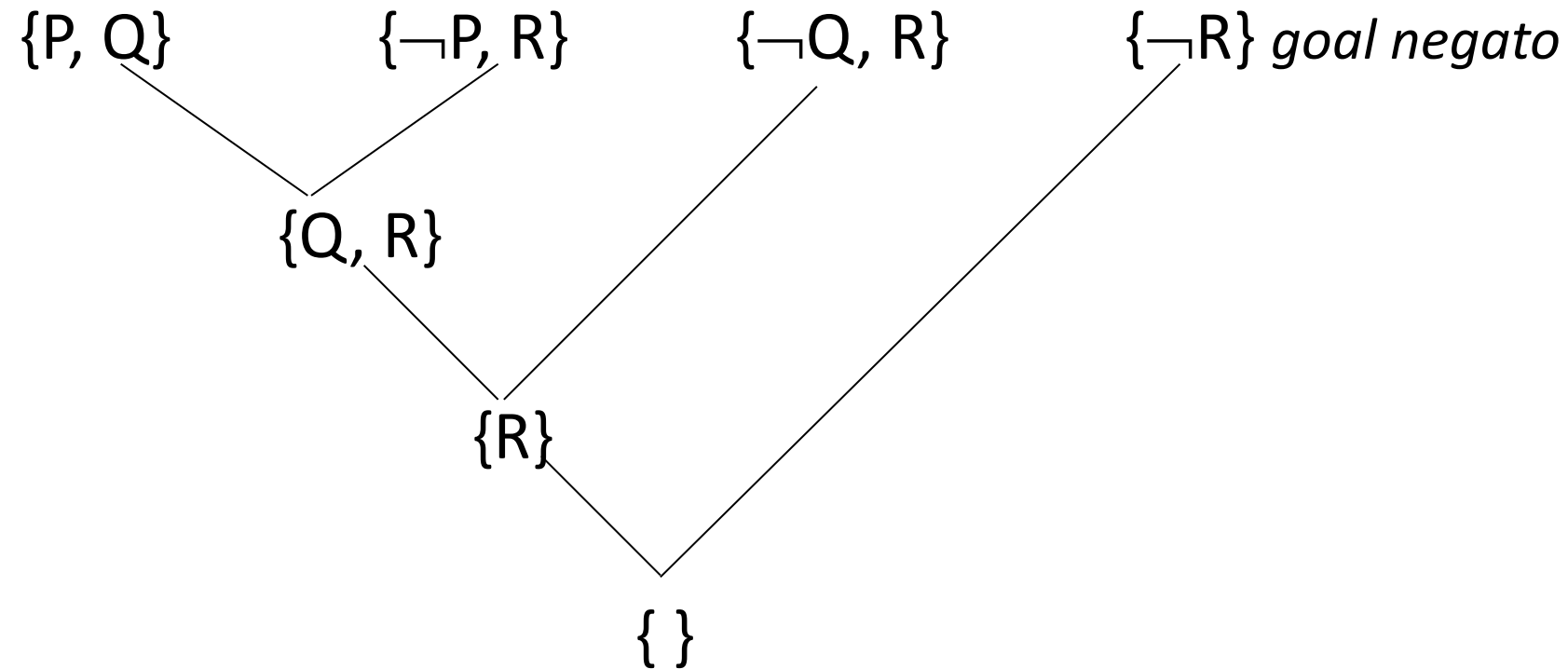


# Risoluzione ordinata

- Ogni clausola è un insieme ordinato di letterali e si possono unificare solo i primi letterali 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:

$$\neg P_1 \vee \dots \vee \neg P_k \vee Q$$

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

- Una KB a regole

$$P_1 \wedge \dots \wedge P_k \Rightarrow Q \text{ regola}$$

$$Q \text{ fatto}$$



# Sistemi a regole logici

- Se la KB contiene solo clausole Horn *definite* i meccanismi inferenziali sono molto più semplici, il processo molto più “guidato”, senza rinunciare alla completezza.
- Risolutori in tempo lineare per il caso proposizionale
- 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-consequente”
  - 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

$A.$  fatto

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

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

Se  $B_1 \wedge B_2 \wedge \dots \wedge B_n$

è il goal

$\neg(B_1 \wedge B_2 \wedge \dots \wedge B_n) \vee \text{False}$

è il goal negato, ovvero

$B_1 \wedge B_2 \wedge \dots \wedge B_n \Rightarrow \text{False}$

che viene scritto

$:- B_1, B_2, \dots, B_n$

omettendo il conseguente

# Programmi logici

- Interpretazione dichiarativa di una regola

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

A è vero se sono veri  $B_1, B_2, \dots, B_n$

In accordo al significato logico dell'implicazione.

- Interpretazione procedurale

la testa può essere vista come una chiamata di procedura e il corpo come una serie di procedure da eseguire in sequenza

# 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.

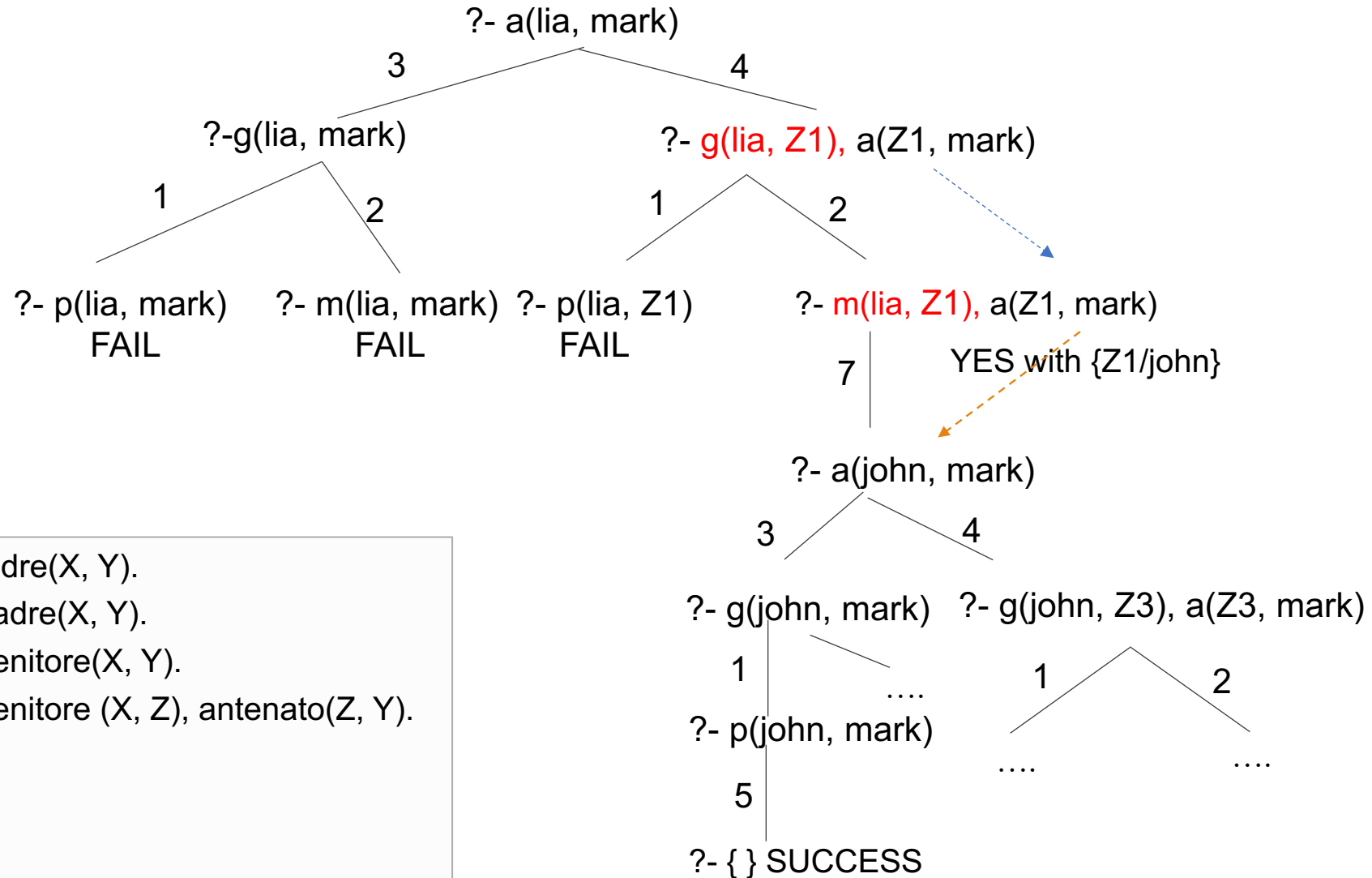
# Alberi di risoluzione SLD

- Dato un programma logico  $P$ , l'albero SLD per un goal  $G$  è definito come segue:
  - ogni nodo dell'albero corrisponde a un goal [coniuntivo]
  - la radice è  $\text{:- } G_1, G_2, \dots, G_k$ , il nostro goal
  - sia  $\text{:- } G_1, G_2, \dots, G_k$  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  $A \text{ :- } B_1, \dots, B_k$  e  $A$  è unificabile con  $G_1$  e  $\gamma = \text{MGU}(A, G_1)$   
un discendente è il goal  $\text{:- } (B_1, \dots, B_k, G_2, \dots, G_k)\gamma$
  - 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 il goal *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.  $padre(john, mark).$
6.  $padre(john, luc).$
7.  $madre(lia, john).$



# Risoluzione SLD

- La strategia è completa per clausole Horn definite
  - 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 and.
- La sostituzione corrispondente è la *risposta calcolata*

# Strategia di visita dell'albero SLD e PROLOG

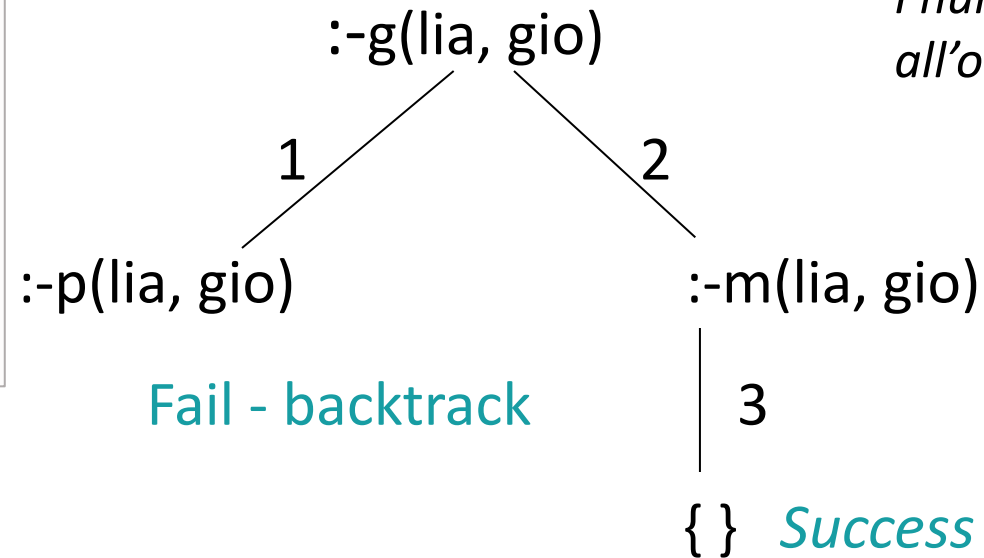
- A seconda di come l'albero viene visitato si potrebbe anche non trovare la clausola vuota: la strategia di ricerca può essere responsabile dell'incompletezza.
- In PROLOG, il più famoso linguaggio di programmazione logica, la visita dell'albero di risoluzione avviene con una ricerca in profondità,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

# 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*

# PROLOG e domande del tipo “si-no”

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(john, mark).
6. padre(john, luc).
7. madre(lia, john).



*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”

$\text{:- } p(X, \text{mark})$

*chi è il padre di Mark?*

$X = \text{gio}$

$\text{:- } p(X, \text{mark})$

1

{ }

con {X/gio}

$\text{:- } p(\text{gio}, X)$

*chi sono i figli di Gio?*

$X = \text{mark};$

$X = \text{luc}.$

$\text{:- } p(\text{gio}, X)$

1

{ }

con {X/mark}

2

{ }

con {X/luc}

## Altre domande ...

- Chi è figlio di chi?  
:-  $g(X, Y)$ .
- Trova 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.  $g(X, Y) \text{ :- } p(X, Y)$
2.  $g(X, Y) \text{ :- } m(X, Y)$
4.  $a(X, Y) \text{ :- } a(Z, Y), g(X, Z)$
3.  $a(X, Y) \text{ :- } g(X, Y)$
5.  $p(\text{gio}, \text{mark})$
6.  $p(\text{gio}, \text{luc})$
7.  $m(\text{lia}, \text{gio})$

*Goal:*

$\text{:- } a(\text{lia}, \text{mark})$   
 $\text{:- } a(Z_1, \text{mark}), a(\text{lia}, Z_1)$   
 $\text{:- } a(Z_2, \text{mark}), g(Z_1, Z_2), g(\text{lia}, Z_1)$   
 $\text{:- } a(Z_3, \text{mark}), g(Z_2, Z_3), g(Z_1, Z_2), g(\text{lia}, Z_1)$   
...

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

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

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



# 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:  $>$ ,  $<$ ,  $>=$ ,  $<=$ ,  $:=$ ,  $\backslash=$  forzano la valutazione, variabili ok purché istanziate

Nota:  $2+1 = 1+2$  unificazione fallisce;  $2+1 := 1+2$  ok

- Esempio:

$\text{max}(X, Y, Y) :- X \leq Y.$

$\text{max}(X, Y, X) :- X > Y.$

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/>

- SWISH Prolog online

<http://swish.swi-prolog.org/>

# Sistemi a regole in avanti

- *Modus ponens generalizzato*

$$\frac{p_1' \ p_2' \ \dots \ p_n' \quad (p_1 \wedge p_2 \wedge \dots \wedge p_n \Rightarrow q)}{(q) \ \theta}$$

dove  $\theta = \text{MGU}(p_i', p_i)$ , per ogni  $i$

- Regola corretta:
  - Si istanziano gli universali
  - Si istanziano le regole
  - Si applica il Modus Ponens classico
- Più generale di MP, ma anche più limitata nella forma del conseguente

# Esempio di MP generalizzato

- Supponiamo di avere nella KB:

King(John)

Greedy(y)

$\text{King}(x) \wedge \text{Greedy}(x) \Rightarrow \text{Evil}(x)$

Con  $\theta = \{x/\text{John}, y/\text{John}\}$  si ottiene

$\text{King}(\text{John}), \text{Greedy}(\text{John}), \text{King}(\text{John}) \wedge \text{Greedy}(\text{John}) \Rightarrow \text{Evil}(\text{John})$

e quindi la conclusione della regola è

$\text{Evil}(\text{John})$

# Esempio di concatenazione in avanti

*È un crimine per un Americano vendere armi a una nazione ostile. Il paese Nono, un nemico dell'America, ha dei missili, e tutti i missili gli sono stati venduti dal colonnello West, un Americano.*

Dimostrare: che *West* è un criminale

# Formalizzazione

1.  $\text{Americano}(x) \wedge \text{Arma}(y) \wedge \text{Vende}(x, y, z) \wedge \text{Ostile}(z) \Rightarrow \text{Criminale}(x)$
2.  $\exists x \text{Possiede}(\text{Nono}, x) \wedge \text{Missile}(x)$   
 $\text{Possiede}(\text{Nono}, M_1) \wedge \text{Missile}(M_1)$
3.  $\text{Missile}(x) \wedge \text{Possiede}(\text{Nono}, x) \Rightarrow \text{Vende}(\text{West}, x, \text{Nono})$
4.  $\text{Missile}(x) \Rightarrow \text{Arma}(x)$
5.  $\text{Nemico}(x, \text{America}) \Rightarrow \text{Ostile}(x)$
6.  $\text{Americano}(\text{West})$
7.  $\text{Nemico}(\text{Nono}, \text{America})$

# Concatenazione in avanti

- Un semplice processo inferenziale (FOL\_FC\_Ask) applica ripetutamente il Modus Ponens generalizzato per ottenere nuovi fatti fino a che
  - si dimostra quello che si desidera
  - nessun fatto nuovo può essere aggiunto
- Una strategia di ricerca sistematica in ampiezza
- In questo caso non ci sono funzioni e il processo converge: siamo nelle condizioni di un database Datalog (basi di dati deduttive)

# Concatenazione in avanti: esempio

I iterazione:

2.  $\text{Possiede}(\text{Nono}, M_1) \wedge \text{Missile}(M_1)$
3.  $\text{Missile}(x) \wedge \text{Possiede}(\text{Nono}, x) \Rightarrow \text{Vende}(\text{West}, x, \text{Nono})$ 
  - La regola 3 è soddisfatta con  $\{x/M_1\}$  e viene aggiunto
  - $\text{Vende}(\text{West}, M_1, \text{Nono})$
4.  $\text{Missile}(x) \Rightarrow \text{Arma}(x)$ 
  - La regola 4 è soddisfatta con  $\{x/M_1\}$  e viene aggiunto
  - $\text{Arma}(M_1)$
5.  $\text{Nemico}(x, \text{America}) \Rightarrow \text{Ostile}(x)$
6.  $\text{Nemico}(\text{Nono}, \text{America})$ 
  - La regola 5 è soddisfatta con  $\{x/\text{Nono}\}$  e viene aggiunto
  - $\text{Ostile}(\text{Nono})$

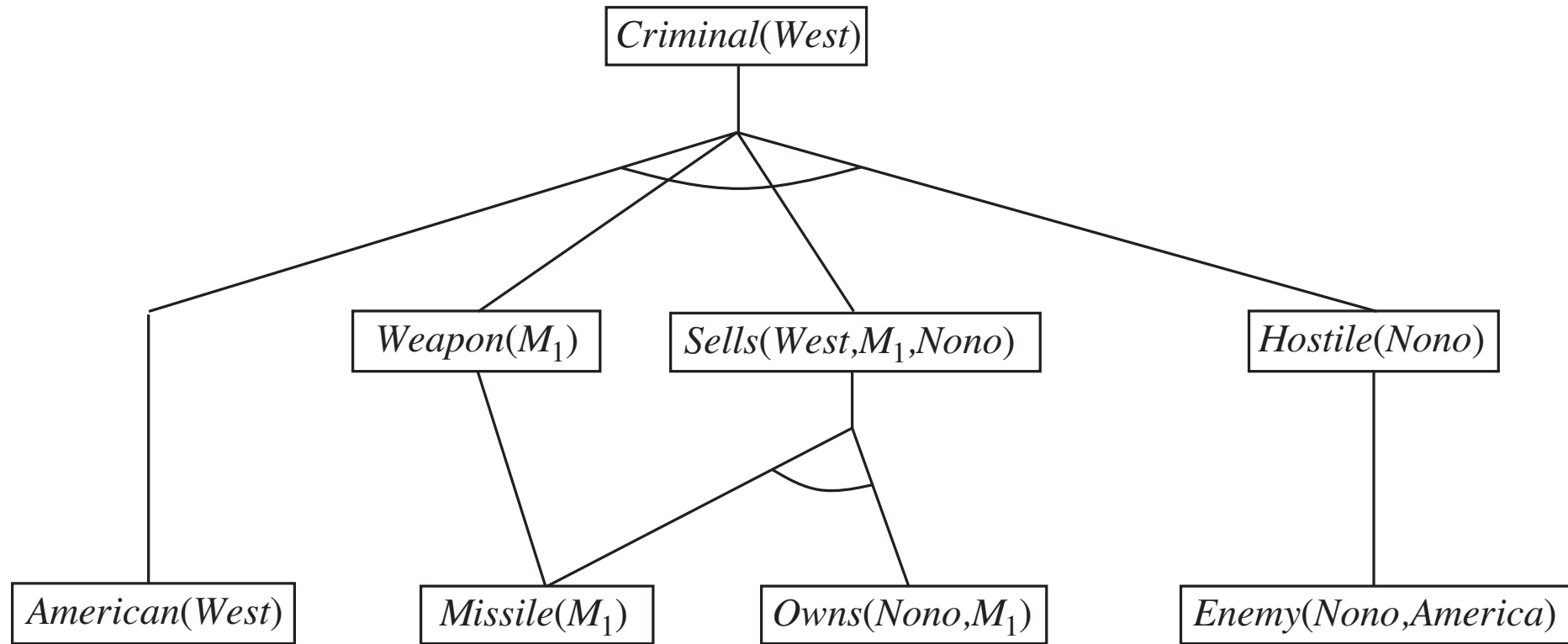


# Concatenazione in avanti: esempio

## Il iterazione

1.  $\text{Americano}(x) \wedge \text{Arma}(y) \wedge \text{Vende}(x, y, z) \wedge \text{Ostile}(z) \Rightarrow \text{Criminale}(x)$ 
  - La regola 1 è soddisfatta con  $\{x/\text{West}, y/M_1, z/\text{Nono}\}$
  - $\text{Criminale}(\text{West})$  viene aggiunto.

# La dimostrazione in avanti



Nota: si parte dal basso

# Analisi di FOL-FC-Ask

- Corretta perché il MP generalizzato è corretto
- Completa per KB di clausole Horn definite
  - Completa e convergente per calcolo proposizionale e per KB di tipo DATALOG (senza funzioni) perché la chiusura deduttiva è un insieme finito
  - Completa anche con funzioni ma il processo potrebbe non terminare (semidecidibile)
- Il metodo descritto è sistematico ma non troppo efficiente

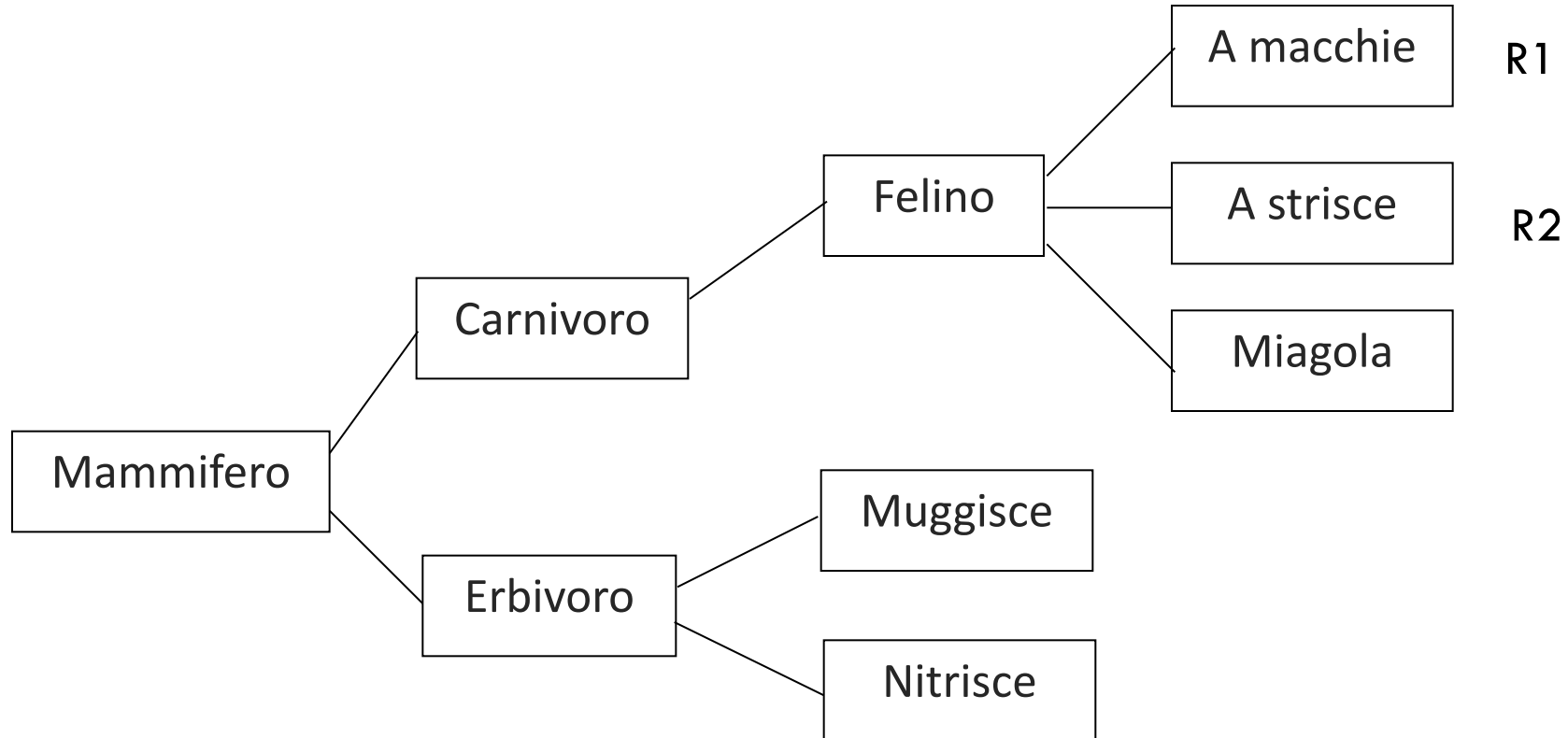
# FC efficiente

- Ordinamento dei congiunti:  
conviene soddisfare prima i congiunti con meno istanze nella KB e che compaiono in più regole  
$$\text{Missile}(x) \wedge \text{Possiede}(\text{Nono}, x) \Rightarrow \text{Vende}(\text{West}, x, \text{Nono})$$
  
Tipi di missile << cose possedute
- Altre ottimizzazioni mutuuate dai CSP

# FC efficiente: rete di discriminazione

- *Assunzione 1*: regole diverse possono condividere molte delle precondizioni. Esempio:  
R1: (Mammifero x) (Felino x) (Carnivoro x)  
    (A-Macchie x)  $\Rightarrow$  (assert (Leopardo x))  
R2: (Mammifero x) (Felino x) (Carnivoro x)  
    (A-Strisce x)  $\Rightarrow$  (assert (Tigre x))
- *Idea* : codificare gli antecedenti delle regole sotto forma di *rete di discriminazione*

# Esempio di rete di discriminazione



# FC incrementale

- Ogni nuovo fatto inferito al tempo  $t$  deve essere dedotto usando almeno un fatto dedotto al tempo  $t-1$
- si possono guardare solo le regole che hanno premesse unificabili con fatti aggiunti nell'ultima iterazione
- indicizzare le regole sui fatti
- evitare di ricalcolare le unificazioni
- altre ottimizzazioni presenti nell'algoritmo RETE ...

# FC efficiente: ridurre deduzioni irrilevanti

- Un modo per evitare di ricavare fatti irrilevanti
- Lavorando all'indietro dal goal, non c'è questo problema
- Si fa una specie di pre-processing per individuare le regole che servono, procedendo all'indietro dal goal e marcando le regole utili.
- Poi si procede in avanti utilizzando solo le regole marcate.



# FC efficiente: l'idea del *magic set*

- Goal: Criminal(West)  $KB \leftarrow KB \cup \{\text{Magic(West)}\}$
- Riscrittura regole:
  - $\text{Magic}(x) \wedge \text{Americano}(x) \wedge \text{Arma}(y)$   
 $\wedge \text{Vende}(x, y, z) \wedge \text{Ostile}(z) \Rightarrow \text{Criminale}(x)$
- Procedendo poi in avanti saranno utilizzate solo le “regole magiche” in modo mirato.
- Combina BC e FC

# Conclusioni

- KB logici in forma a regole
- Sistemi a regole in avanti (sistemi di produzione) sono usati per la costruzione di sistemi esperti (CLIPS)
- Conoscenza espressa naturalmente in forma di regole. Conseguente può essere una qualunque azione. Paradigma di programmazione generale
- Possibilità di giustificazione