Agenti logici: sistemi a regole, cenni

Paolo Mancarella

a.a. 2022-2023

Il sottoinsieme "a regole" del FOL

- *Clausole Horn definite: esattamente un letterale positivo*

  $\{Q, \neg P_1, \dots, \neg P_k\}$ (k≥0)

- 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

  $P_1 \wedge \dots \wedge P_k \Rightarrow Q$ *regola* k>0

  Q *fatto* k=0

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*): ragionamento guidato dall’obiettivo
  - Le regole sono applicate alla rovescia
  - Programmazione logica (PROLOG)
- Concatenazione in avanti (*Forward Chaining*): 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.	fatto
$A :- B_1, B_2, \ldots, B_n.$	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 $B_1 \wedge B_2 \wedge \ldots \wedge B_n$	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

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 $B_1, B_2, \ldots B_n$

  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

Esempio di programma logico

```
genitore(X, Y) :– padre(X, Y).
genitore(X, Y) :– madre(X, Y).
antenato(X, Y) :– genitore(X, Y).
antenato(X, Y) :– genitore(X, Z), antenato(Z, Y).
padre(gio, mark).
padre(gio, luc).
madre(lia, gio).

:– antenato(lia, mark).      goal (negato)
```

Risoluzione SLD

- La risoluzione SLD (Selection Linear Definite-clauses) è una strategia *ordinata*
- La risoluzione SLD è completa per clausole Horn
- A partire da un programma P e da un goal G si costruisce l’albero di risoluzione

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_1, G_2, … , G_k, il goal di partenza
  - sia :– G_1, G_2, … , 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, … , B_k, e A è unificabile con G_1 con γ = MGU(A, G_1)

    un discendente è il goal :– (B_1, … , B_k, G_2, … , G_k)γ
  - i nodi che sono clausole vuote sono *successi*
  - i nodi che non hanno successori sono *fallimenti*

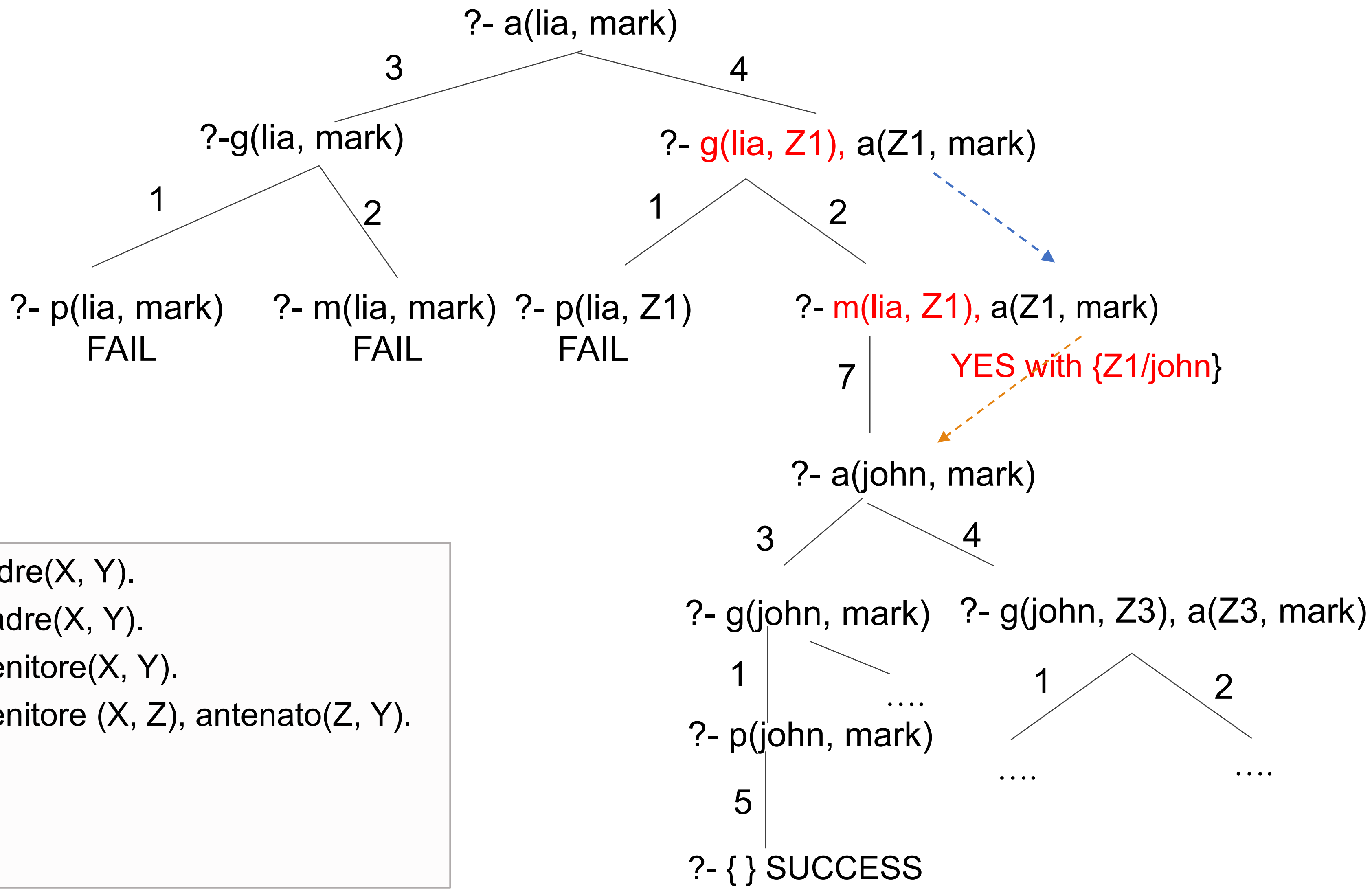
Esempio di albero SLD: il programma

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).

Ci chiediamo se *lia* è un antenato di *mark*

:– 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.
- 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

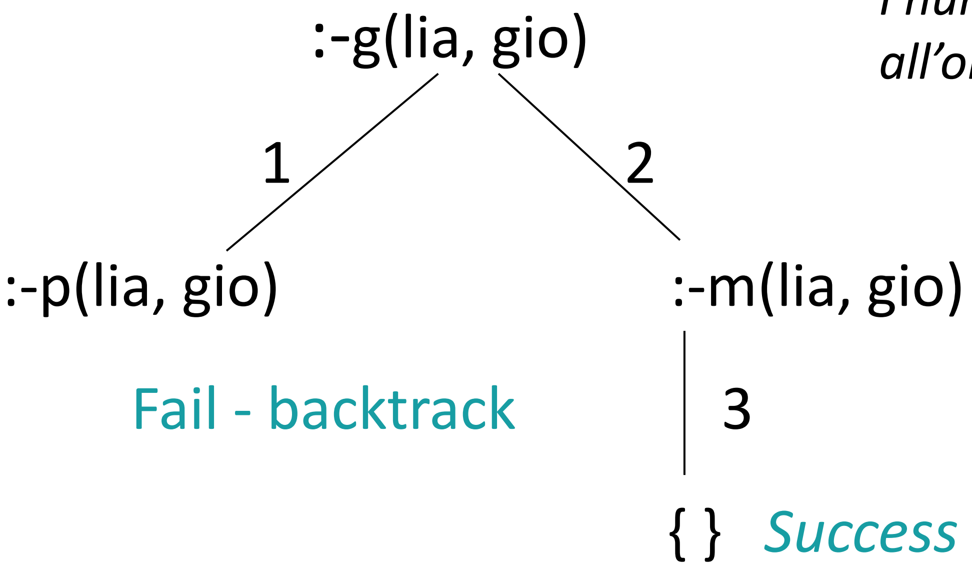
1. genitore(X, Y) :– padre(X, Y).
2. genitore(X, Y) :– madre(X, Y).
3. antenato(X, Y) :– genitore(X, Y).
4. antenato(X, Y) :– genitore(X, Z), antenato(Z, Y).
5. padre(gio, mark).
6. padre(gio, luc).
7. madre(lia, gio).

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

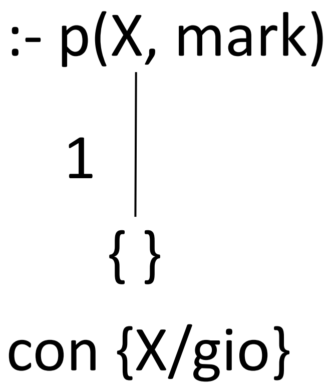
Assunzione di mondo chiuso

PROLOG con domande del tipo “trova”

:- p(X, mark)

chi è il padre di Mark?

X = gio

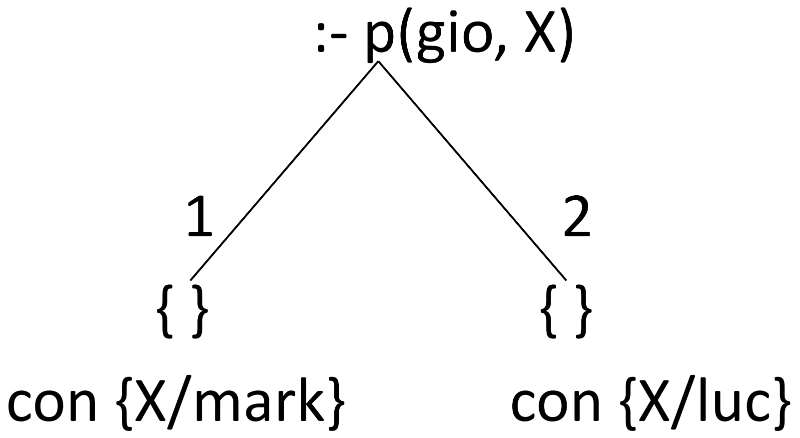


:- p(gio, X)

chi sono i figli di Gio?

X = mark;

X = luc.



Altre domande ...

- Chi è figlio di chi?

  :- g(X, Y).
- 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) :- 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), a(lia, Z_1)

:- a(Z_2, mark), g(Z_1, Z_2), g(lia, Z_1)

:- a(Z_3, mark), g(Z_2, Z_3), g(Z_1, Z_2), g(lia, Z_1)

...

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:

```
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 5.
- 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/

- SWISH Prolog online

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

Sistemi a regole in avanti

- *Modus ponens generalizzato*

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

  dove $\theta = \mathrm{MGU}(p_i', p_i)$, 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 (la premessa è una congiunzione di formule atomiche, non una formula qualunque)

Esempio di MP generalizzato

- Supponiamo di avere nella KB:

  King(John)

  Greedy(y)

  King(x) $\wedge$ Greedy(x) $\Rightarrow$ Evil(x)

 Con θ={x/John, y/John} si ottiene

 King(John), Greedy(John), King(John)$\wedge$Greedy(John) $\Rightarrow$ Evil(John)

 e quindi la conclusione della regola è

  Evil(John)

Esempio di concatenazione in avanti

È un crimine per un Ameri cano 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. $\exists x\ \text{Possiede}(\text{Nono}, x) \wedge \text{Missile}(x)$

   $\text{Possiede}(\text{Nono}, M_1) \wedge \text{Missile}(M_1)$ *(skolemizzazione)*
3. $\text{Missile}(x) \wedge \text{Possiede}(\text{Nono}, x) \Rightarrow \text{Vende}(\text{West}, x, \text{Nono})$
4. $\text{Missile}(x) \Rightarrow \text{Arma}(x)$ *common sense*
5. $\text{Nemico}(x, \text{America}) \Rightarrow \text{Ostile}(x)$ *common sense*
6. $\text{Americano}(\text{West})$
7. $\text{Nemico}(\text{Nono}, \text{America})$

Concatenazione in avanti

- Un semplice processo inferenziale (FOL_CA_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

Prima iterazione:

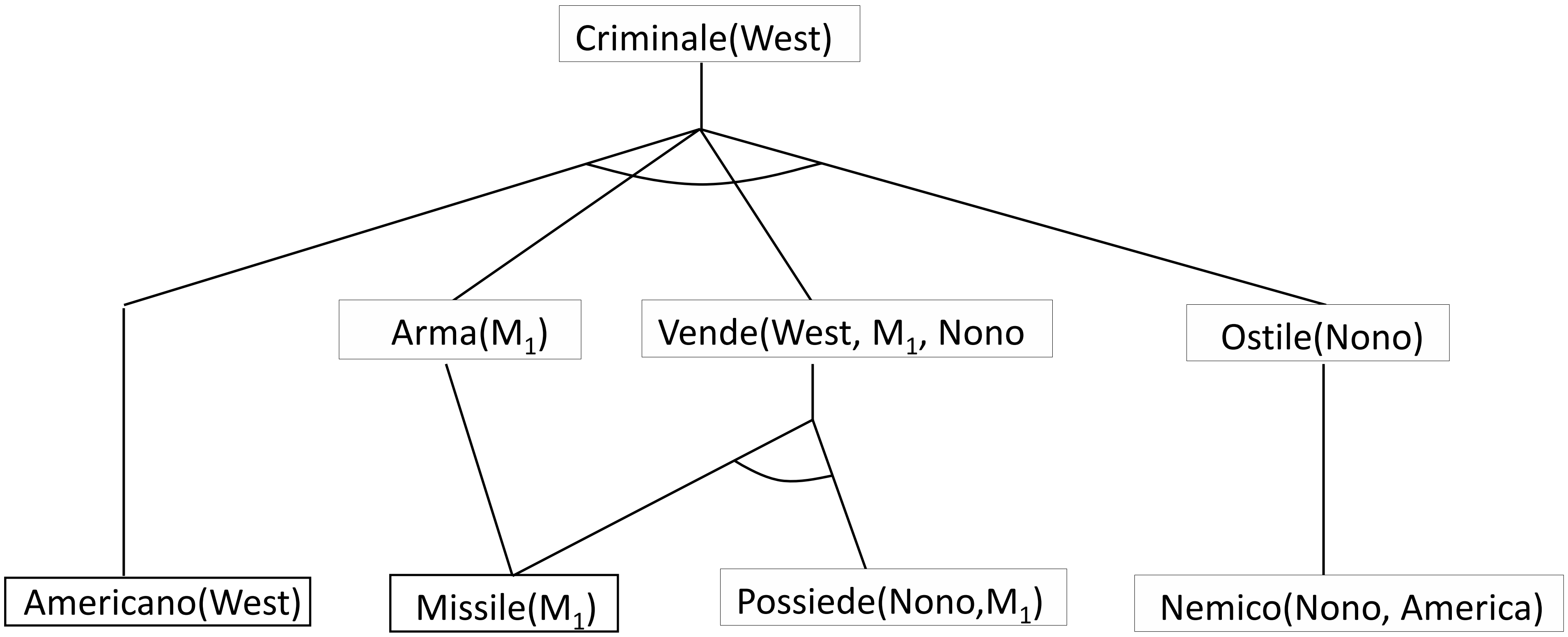
2. Possiede(Nono, M_1) $\wedge$ Missile(M_1)
3. Missile(x)$\wedge$Possiede(Nono,x) $\Rightarrow$ Vende(West,x,Nono)
   - La regola 3 è soddisfatta con {x/M_1} e viene aggiunto
   - Vende(West, M_1, Nono)
4. Missile(x) $\Rightarrow$ Arma(x)
   4. La regola 4 è soddisfatta con {x/M_1} e viene aggiunto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

Concatenazione in avanti: esempio

Seconda iterazione

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