Inferenza nella Logica del Primo Ordine

Capitolo 9

Outline

- ◇ Riduzione della inferenza di primo ordine alla inferenza proposizionale
- ◇ Unificazione
- ◇ Modus Ponens generalizzato
- ◇ Forward e backward chaining
- ◇ Programmazione Logica
- ◇ Risoluzione

Istanziazione Universale (UI)

Ogni istanziazione di una sentenza quantificata universalmente è conseguenza logica di quest'ultima:

$$\frac{\forall v \;\; \alpha}{\textsc{Subst}(\{v/g\}, \alpha)}$$

per ogni variabile v e termine ground g

P.e., $\forall x \;\; Re(x) \land Ingordo(x) \;\Rightarrow\; Diavolo(x)$ porta a

$$\begin{array}{l} Re(Giovanni) \land Ingordo(Giovanni) \;\Rightarrow\; Diavolo(Giovanni) \\ Re(Riccardo) \land Ingordo(Riccardo) \;\Rightarrow\; Diavolo(Riccardo) \\ Re(Padre(Giovanni)) \land Ingordo(Padre(Giovanni)) \;\Rightarrow\; Diavolo(Padre(Giovanni)) \\ \quad\vdots \end{array}$$

Istanziazione Esistenziale (EI)

Per ogni sentenza α, variabile v, e simbolo costante k *che non appare in nessuna parte della base di conoscenza*:

$$\frac{\exists v \;\; \alpha}{\textsc{Subst}(\{v/k\}, \alpha)}$$

P.e., $\exists x \;\; Corona(x) \land SullaTesta(x, Giovanni)$ porta a

$$Corona(C_1) \land SullaTesta(C_1, Giovanni)$$

a patto che C_1 sia un nuovo simbolo di costante, chiamato costante di Skolem

Altro esempio: da $\exists x \;\; d(x^y)/dy = x^y$ si ottiene

$$d(e^y)/dy = e^y$$

a patto che e sia un nuovo simbolo di costante

Istanziazione Esistenziale

UI può essere applicata più volte per *aggiungere* nuove sentenze;
la nuova KB è logicamente equivalente alla vecchia

EI può essere applicata solo una volta per *rimpiazzare* la sentenza esistenziale;
la nuova KB *non* è equivalente alla vecchia,
ma è soddisfacibile sse la vecchia KB era soddisfacibile

Riduzione alla inferenza proposizionale

Supponiamo che KB contenga solo le seguenti sentenze:

$$\forall x \;\; Re(x) \land Ingordo(x) \;\Rightarrow\; Diavolo(x)$$
$$Re(Giovanni)$$
$$Ingordo(Giovanni)$$
$$Fratello(Riccardo, Giovanni)$$

Istanziando la sentenza universale in *tutti i possibili* modi, si ottiene

$$Re(Giovanni) \land Ingordo(Giovanni) \;\Rightarrow\; Diavolo(Giovanni)$$
$$Re(Riccardo) \land Ingordo(Riccardo) \;\Rightarrow\; Diavolo(Riccardo)$$
$$Re(Giovanni)$$
$$Ingordo(Giovanni)$$
$$Fratello(Riccardo, Giovanni)$$

La nuova KB è proposizionalizzata: i simboli proposizionali sono

$Re(Giovanni)$, $Ingordo(Giovanni)$, $Diavolo(Giovanni)$, $Re(Riccardo)$ etc.

Riduzione

Affermazione: una sentenza ground è conseguenza logica della nuova KB sse è conseguenza logica della KB originaria

Affermazione: ogni FOL KB può essere proposizionalizzata in modo da preservarne le conseguenze logiche

Idea: proposizionalizzare sia KB che la query, applicare la risoluzione, restituire il risultato

Problema: con i simboli di funzione, c'è un numero infinito di termini ground,
e.g., $Padre(Padre(Padre(Giovanni)))$

Teorema: Herbrand (1930). Se una sentenza α è conseguenza logica di una FOL KB,
essa è conseguenza logica di un sottoinsieme *finito* della KB proposizionale

Idea: For $n = 0$ to ∞ do
creare una KB proposizionale con termini di profondità n
controllare se α è conseguenza logica della KB considerata

Problema: funziona solo se α è conseguenza logica, cicla se α non è conseguenza logica

Teorema: Turing (1936), Church (1936), "entailment" in FOL è semidecidible

Problemi con la proposizionalizzazione

La proposizionalizzazione sembra generare tante sentenze irrilevanti
P.e.,

$$\forall x \;\; Re(x) \land Ingordo(x) \;\Rightarrow\; Diavolo(x)$$
$$Re(Giovanni)$$
$$\forall y \;\; Ingordo(y)$$
$$Fratello(Riccardo, Giovanni)$$

ok per $Diavolo(Giovanni)$, ma la proposizionalizzazione produce tanti fatti come $Ingordo(Riccardo)$ che sono irrilevanti

Con p predicati k-ari e n constanti, ci sono $p \cdot n^k$ istanziazioni!

Unificazione

Si può ottenere l'inferenza immediatamente se possiamo trovare una sostituzione θ tale che $Re(x)$ e $Ingordo(x)$ corrispondano a $Re(Giovanni)$ e $Ingordo(y)$

$\theta = \{x/Giovanni, y/Giovanni\}$ va bene

$\text{UNIFY}(\alpha, \beta) = \theta$ if $\alpha\theta = \beta\theta$

p	q	θ
$Conosce(Giovanni, x)$	$Conosce(Giovanni, Giovanna)$	
$Conosce(Giovanni, x)$	$Conosce(y, OJ)$	
$Conosce(Giovanni, x)$	$Conosce(y, Madre(y))$	
$Conosce(Giovanni, x)$	$Conosce(x, OJ)$	

Unificazione

Si può ottenere l'inferenza immediatamente se possiamo trovare una sostituzione θ tale che $Re(x)$ e $Ingordo(x)$ corrispondano a $Re(Giovanni)$ e $Ingordo(y)$

$\theta = \{x/Giovanni, y/Giovanni\}$ va bene

$\text{UNIFY}(\alpha, \beta) = \theta$ if $\alpha\theta = \beta\theta$

p	q	θ
$Conosce(Giovanni, x)$	$Conosce(Giovanni, Giovanna)$	$\{x/Giovanna\}$
$Conosce(Giovanni, x)$	$Conosce(y, OJ)$	
$Conosce(Giovanni, x)$	$Conosce(y, Madre(y))$	
$Conosce(Giovanni, x)$	$Conosce(x, OJ)$	

Unificazione

Si può ottenere l'inferenza immediatamente se possiamo trovare una sostituzione θ tale che $Re(x)$ e $Ingordo(x)$ corrispondano a $Re(Giovanni)$ e $Ingordo(y)$

$\theta = \{x/Giovanni, y/Giovanni\}$ va bene

$\text{UNIFY}(\alpha, \beta) = \theta$ if $\alpha\theta = \beta\theta$

p	q	θ
$Conosce(Giovanni, x)$	$Conosce(Giovanni, Giovanna)$	$\{x/Giovanna\}$
$Conosce(Giovanni, x)$	$Conosce(y, OJ)$	$\{x/OJ, y/Giovanni\}$
$Conosce(Giovanni, x)$	$Conosce(y, Madre(y))$	
$Conosce(Giovanni, x)$	$Conosce(x, OJ)$	

Unificazione

Si può ottenere l'inferenza immediatamente se possiamo trovare una sostituzione θ tale che $Re(x)$ e $Ingordo(x)$ corrispondano a $Re(Giovanni)$ e $Ingordo(y)$

$\theta = \{x/Giovanni, y/Giovanni\}$ va bene

$\text{UNIFY}(\alpha, \beta) = \theta$ if $\alpha\theta = \beta\theta$

p	q	θ
$Conosce(Giovanni, x)$	$Conosce(Giovanni, Giovanna)$	$\{x/Giovanna\}$
$Conosce(Giovanni, x)$	$Conosce(y, OJ)$	$\{x/OJ, y/Giovanni\}$
$Conosce(Giovanni, x)$	$Conosce(y, Madre(y))$	$\{y/Giovanni, x/Madre(Giovanni)\}$
$Conosce(Giovanni, x)$	$Conosce(x, OJ)$	

Unificazione

Si può ottenere l'inferenza immediatamente se possiamo trovare una sostituzione θ tale che $Re(x)$ e $Ingordo(x)$ corrispondano a $Re(Giovanni)$ e $Ingordo(y)$

$\theta = \{x/Giovanni, y/Giovanni\}$ va bene

$\text{UNIFY}(\alpha, \beta) = \theta$ if $\alpha\theta = \beta\theta$

p	q	θ
$Conosce(Giovanni, x)$	$Conosce(Giovanni, Giovanna)$	$\{x/Giovanna\}$
$Conosce(Giovanni, x)$	$Conosce(y, OJ)$	$\{x/OJ, y/Giovanni\}$
$Conosce(Giovanni, x)$	$Conosce(y, Madre(y))$	$\{y/Giovanni, x/Madre(Giovanni)\}$
$Conosce(Giovanni, x)$	$Conosce(x, OJ)$	$fallimento$

Standardizing apart elimina sovrapposizioni di variabili, p.e., $Conosce(z_{17}, OJ)$

Unificazione

Si può ottenere l'inferenza immediatamente se possiamo trovare una sostituzione θ tale che $Re(x)$ e $Ingordo(x)$ corrispondano a $Re(Giovanni)$ e $Ingordo(y)$

$\theta = \{x/Giovanni, y/Giovanni\}$ va bene

$\text{UNIFY}(\alpha, \beta) = \theta$ if $\alpha\theta = \beta\theta$

p	q	θ
$Conosce(Giovanni, x)$	$Conosce(Giovanni, Giovanna)$	$\{x/Giovanna\}$
$Conosce(Giovanni, x)$	$Conosce(y, OJ)$	$\{x/OJ, y/Giovanni\}$
$Conosce(Giovanni, x)$	$Conosce(y, Madre(y))$	$\{y/Giovanni, x/Madre(Giovanni)\}$
$Conosce(Giovanni, x)$	$Conosce(x, OJ)$	$fallimento$

Standardizing apart elimina sovrapposizioni di variabili, p.e., $Conosce(z_{17}, OJ)$

$Conosce(Giovanni, x)$	$Conosce(z_{17}, OJ)$	$\{z_{17}/Giovanni, x/OJ\}$

Algoritmo di Unificazione

```
function UNIFY(x, y, θ) returns a substitution to make x and y identical (given θ)
   inputs: x, a variable, constant, list, or compound
           y, a variable, constant, list, or compound
           θ, the substitution built up so far, initially empty

   if θ = failure then return failure
   else if x = y then return θ
   else if VARIABLE?(x) then return UNIFY-VAR(x, y, θ)
   else if VARIABLE?(y) then return UNIFY-VAR(y, x, θ)
   else if COMPOUND?(x) and COMPOUND?(y) then
       return UNIFY(ARGS[x], ARGS[y], UNIFY(OP[x], OP[y], θ))
   else if LIST?(x) and LIST?(y) then
       return UNIFY(REST[x], REST[y], UNIFY(FIRST[x], FIRST[y], θ))
   else return failure

function UNIFY-VAR(var, x, θ) returns a substitution
   inputs: var, a variable, x any expression, θ the substitution built up so far

   if {var/val} ∈ θ then return UNIFY(val, x, θ)
   else if {x/val} ∈ θ then return UNIFY(var, val, θ)
   else if OCCUR-CHECK?(var, x) then return failure
   else return add {var/x} to θ
```


Algoritmo di Unificazione Modificato

```
function UNIFY(x, y, θ) returns a substitution to make x and y identical (given θ)
   inputs: x, a variable, constant, list, or compound
           y, a variable, constant, list, or compound
           θ, the substitution built up so far, initially empty

   if θ = failure then return failure
   else if x = y then return θ
   else if VARIABLE?(x) then return UNIFY-VAR(x, y, θ)
   else if VARIABLE?(y) then return UNIFY-VAR(y, x, θ)
   else if COMPOUND?(x) and COMPOUND?(y) then
       return UNIFY(ARGS[x], ARGS[y], UNIFY(OP[x], OP[y], θ))
   else if LIST?(x) and LIST?(y) then
       return UNIFY(SUBST(θ,REST[x]), SUBST(θ,REST[y]), UNIFY(SUBST(θ,FIRST[x]),
                                                       SUBST(θ,FIRST[y]), θ))
   else return failure

function UNIFY-VAR(var, x, θ) returns a substitution
   inputs: var, a variable, x any expression, θ the substitution built up so far

   if {var/val} ∈ θ then return UNIFY(val, x, θ)
   else if {x/val} ∈ θ then return UNIFY(var, val, θ)
   else if OCCUR-CHECK?(var, x) then return failure
   else return COMPOSE({var/x},θ)
```

Modus Ponens Generalizzato (GMP)

$$\frac{p_1{}',\ p_2{}',\ \ldots,\ p_n{}',\ (p_1 \land p_2 \land \ldots \land p_n \Rightarrow q)}{q\theta} \qquad \text{dove } p_i{}'\theta = p_i\theta \text{ per tutte le } i$$

$p_1{}'$ è $Re(Giovanni)$ — p_1 è $Re(x)$
$p_2{}'$ è $Ingordo(y)$ — p_2 è $Ingordo(x)$
θ è $\{x/Giovanni, y/Giovanni\}$ — q è $Diavolo(x)$
$q\theta$ è $Diavolo(Giovanni)$

GMP usato con KB composto di clausole definite (*esattamente* un letterale positivo)
Si assume che tutte le variabili siano quantificate universalmente

Correttezza di GMP

Bisogna mostrare che

$$p_1{}',\ \ldots,\ p_n{}',\ (p_1 \land \ldots \land p_n \Rightarrow q) \models q\theta$$

dato che $p_i{}'\theta = p_i\theta$ per tutte le i

Lemma: per ogni clausola definita p, abbiamo $p \models p\theta$ per mezzo di UI

1. $(p_1 \land \ldots \land p_n \Rightarrow q) \models (p_1 \land \ldots \land p_n \Rightarrow q)\theta = (p_1\theta \land \ldots \land p_n\theta \Rightarrow q\theta)$
2. $p_1{}',\ \ldots,\ p_n{}' \models p_1{}' \land \ldots \land p_n{}' \models p_1{}'\theta \land \ldots \land p_n{}'\theta$
3. da 1. e 2., $q\theta$ segue da Modus Ponens ordinario

Esempio di base di conoscenza

La legge in America dice che è un crimine per un americano vendere armi a nazioni ostili. La nazione Nono, nemico dell'America, possiede alcuni missili, venduti alla nazione dal Colonnello West, che è americano.

Provare che Col. West è un criminale

Esempio di base di conoscenza

... è un crimine per un americano vendere armi a nazioni ostili:

Esempio di base di conoscenza

... è un crimine per un americano vendere armi a nazioni ostili:

$American(x) \land Weapon(y) \land Sells(x,y,z) \land Hostile(z) \Rightarrow Criminal(x)$

Nono ... possiede dei missili, cioè, $\exists\, x\ Owns(Nono, x) \land Missile(x)$:

Esempio di base di conoscenza

... è un crimine per un americano vendere armi a nazioni ostili:

$American(x) \land Weapon(y) \land Sells(x,y,z) \land Hostile(z) \Rightarrow Criminal(x)$

Nono ... possiede dei missili, cioè, $\exists\, x\ Owns(Nono, x) \land Missile(x)$:

$Owns(Nono, M_1) \land Missile(M_1)$

... tutti i suoi missili gli sono stati venduti dal Colonnello West

Esempio di base di conoscenza

... è un crimine per un americano vendere armi a nazioni ostili:

$American(x) \land Weapon(y) \land Sells(x,y,z) \land Hostile(z) \Rightarrow Criminal(x)$

Nono ... possiede dei missili, cioè, $\exists\, x\ Owns(Nono, x) \land Missile(x)$:

$Owns(Nono, M_1) \land Missile(M_1)$

... tutti i suoi missili gli sono stati venduti dal Colonnello West

$\forall\, x\ \ Missile(x) \land Owns(Nono, x)\ \Rightarrow\ Sells(West, x, Nono)$

I missili sono armi:

Esempio di base di conoscenza

... è un crimine per un americano vendere armi a nazioni ostili:

$American(x) \land Weapon(y) \land Sells(x,y,z) \land Hostile(z) \Rightarrow Criminal(x)$

Nono ... possiede dei missili, cioè, $\exists\, x\ Owns(Nono, x) \land Missile(x)$:

$Owns(Nono, M_1) \land Missile(M_1)$

... tutti i suoi missili gli sono stati venduti dal Colonnello West

$\forall\, x\ \ Missile(x) \land Owns(Nono, x)\ \Rightarrow\ Sells(West, x, Nono)$

I missili sono armi:

$Missile(x) \Rightarrow Weapon(x)$

Un nemico dell'America è "ostile":

Esempio di base di conoscenza

. . . è un crimine per un americano vendere armi a nazioni ostili:
$American(x) \wedge Weapon(y) \wedge Sells(x, y, z) \wedge Hostile(z) \Rightarrow Criminal(x)$
Nono . . . possiede dei missili, cioè, $\exists\, x\ Owns(Nono, x) \wedge Missile(x)$:
$Owns(Nono, M_1) \wedge Missile(M_1)$
. . . tutti i suoi missili gli sono stati venduti dal Colonnello West
$\forall\, x\ \ Missile(x) \wedge Owns(Nono, x) \ \Rightarrow\ Sells(West, x, Nono)$
I missili sono armi:
$Missile(x) \Rightarrow Weapon(x)$
Un nemico dell'America è "ostile":
$Enemy(x, America) \ \Rightarrow\ Hostile(x)$
West, che è americano . . .

Esempio di base di conoscenza

. . . è un crimine per un americano vendere armi a nazioni ostili:
$American(x) \wedge Weapon(y) \wedge Sells(x, y, z) \wedge Hostile(z) \Rightarrow Criminal(x)$
Nono . . . possiede dei missili, cioè, $\exists\, x\ Owns(Nono, x) \wedge Missile(x)$:
$Owns(Nono, M_1) \wedge Missile(M_1)$
. . . tutti i suoi missili gli sono stati venduti dal Colonnello West
$\forall\, x\ \ Missile(x) \wedge Owns(Nono, x) \ \Rightarrow\ Sells(West, x, Nono)$
I missili sono armi:
$Missile(x) \Rightarrow Weapon(x)$
Un nemico dell'America è "ostile":
$Enemy(x, America) \ \Rightarrow\ Hostile(x)$
West, che è americano . . .
$American(West)$
La nazione Nono, un nemico dell' America . . .

Esempio di base di conoscenza

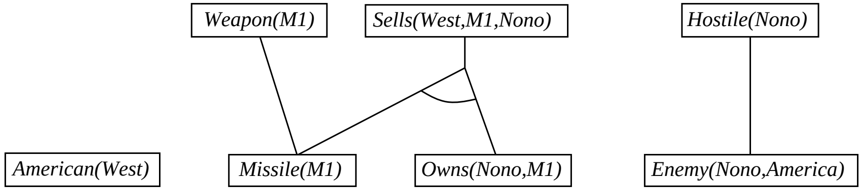
. . . è un crimine per un americano vendere armi a nazioni ostili:
$American(x) \wedge Weapon(y) \wedge Sells(x, y, z) \wedge Hostile(z) \Rightarrow Criminal(x)$
Nono . . . possiede dei missili, cioè, $\exists\, x\ Owns(Nono, x) \wedge Missile(x)$:
$Owns(Nono, M_1) \wedge Missile(M_1)$
. . . tutti i suoi missili gli sono stati venduti dal Colonnello West
$\forall\, x\ \ Missile(x) \wedge Owns(Nono, x) \ \Rightarrow\ Sells(West, x, Nono)$
I missili sono armi:
$Missile(x) \Rightarrow Weapon(x)$
Un nemico dell'America è "ostile":
$Enemy(x, America) \ \Rightarrow\ Hostile(x)$
West, che è americano . . .
$American(West)$
La nazione Nono, un nemico dell' America . . .
$Enemy(Nono, America)$

Algoritmo di Forward chaining

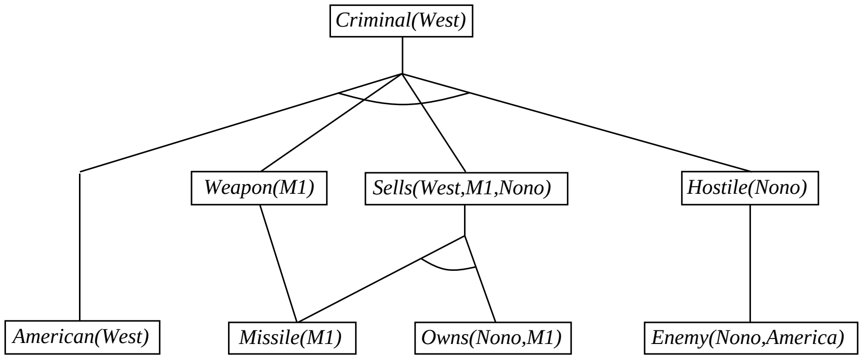
```
function FOL-FC-ASK(KB, α) returns a substitution or false
  repeat until new is empty
      new ← { }
      for each sentence r in KB do
          (p1 ∧ ... ∧ pn ⇒ q) ← STANDARDIZE-APART(r)
          for each θ such that (p1 ∧ ... ∧ pn)θ = (p'1 ∧ ... ∧ p'n)θ
                          for some p'1, ..., p'n in KB
              q' ← SUBST(θ, q)
              if q' is not a renaming of a sentence already in KB or new then do
                  add q' to new
                  φ ← UNIFY(q', α)
                  if φ is not fail then return φ
      add new to KB
  return false
```

Prova tramite Forward chaining

Prova tramite Forward chaining



Prova tramite Forward chaining



Proprietà di Forward chaining

Corretta e completa per clausole definite di primo ordine
(prova simile alla prova proposizionale)

Datalog = clausole definite del primo ordine + *nessuna funzione*
FC termina per Datalog in un numero di iterazioni polinomiale: al più $p \cdot n^k$ letterali

In generale può non terminare se α non è conseguenza logica

Ciò è inevitabile: entailment con clausole definite è semidecidibile

Efficienza di Forward chaining

Semplice osservazione: non c'è bisogno di "match-are" una regola alla iterazione k se una premessa non è stata aggiunta alla iterazione $k-1$
$\Rightarrow$ "match-are" ogni regola le cui premesse contengono un letterale appena aggiunto

Il "matching" è costoso

Indicizzazione della Base di Dati permette il recupero di fatti conosciuti in $O(1)$
p.e., la query $Missile(x)$ recupera $Missile(M_1)$

Matching di premesse congiuntive rispetto a fatti conosciuti è NP-hard

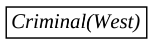
Forward chaining è largamente utilizzato in basi di dati deduttive

Algoritmo Backward chaining

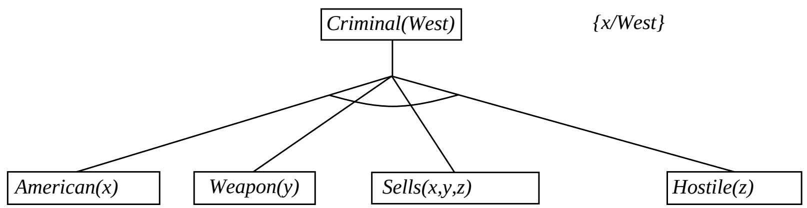
function FOL-BC-ASK(KB, goals, θ) **returns** a set of substitutions
  inputs: KB, a knowledge base
    goals, a list of conjuncts forming a query
    θ, the current substitution, initially the empty substitution $\{\ \}$
  local variables: *ans*, a set of substitutions, initially empty

  if *goals* is empty **then return** $\{\theta\}$
  $q' \leftarrow$ SUBST(θ, FIRST(*goals*))
  for each r **in** KB where STANDARDIZE-APART(r) $= (p_1 \wedge \ldots \wedge p_n \Rightarrow q)$
    and $\theta' \leftarrow$ UNIFY(q, q') succeeds
   ans $\leftarrow$ FOL-BC-ASK(KB, $[p_1, \ldots, p_n|$REST(*goals*)$]$, COMPOSE(θ', θ)) $\cup$ *ans*
  return *ans*

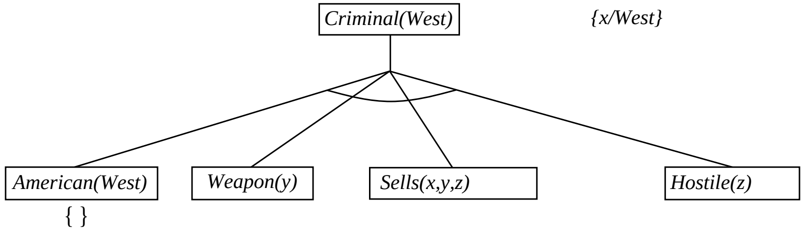
Esempio Backward chaining



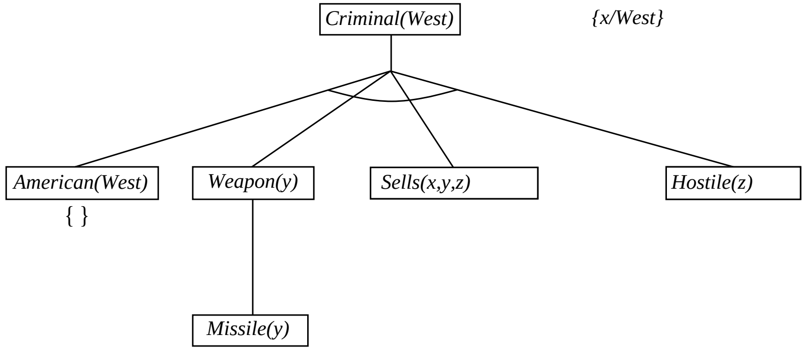
Esempio Backward chaining



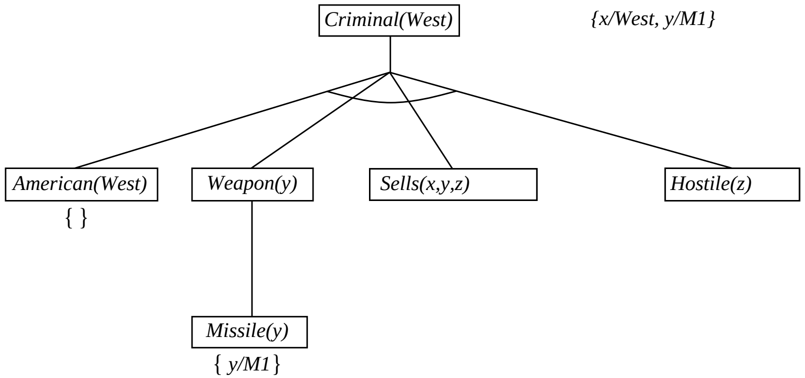
Esempio Backward chaining



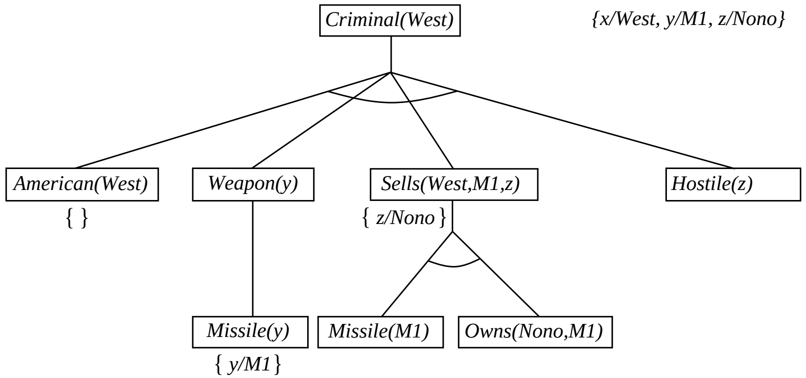
Esempio Backward chaining



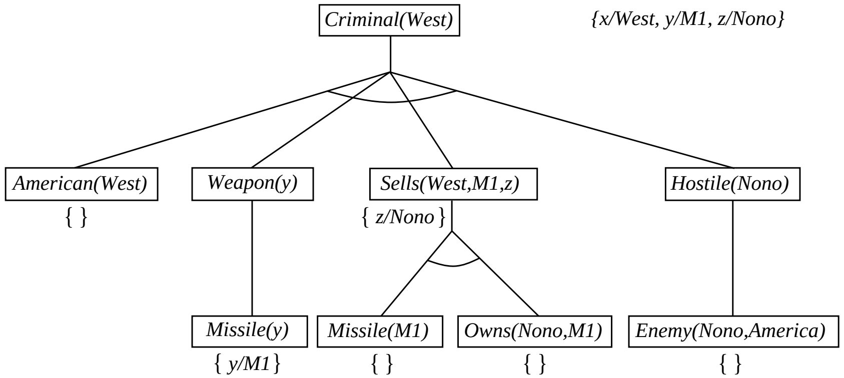
Esempio Backward chaining



Esempio Backward chaining



Esempio Backward chaining



Proprietà di backward chaining

Complessità in spazio lineare con la dimensione della prova

Incompleta a causa di cicli infiniti
⇒ bisogna controllare il goal corrente rispetto ad ogni goal sulla pila che implementa la ricerca in profondità

Inefficiente a causa di sottogoal ripetuti (sia successo che fallimento)
⇒ bisogna usare una cache che contiene i risultati già calcolati (spazio aggiuntivo!)

Largamente utilizzato (senza i miglioramenti!) per la programmazione logica

Programmazione Logica: Prolog

Base: backward chaining con clausole Horn + altro

Tecniche di compilazione ⇒ 60 milioni di LIPS

Programma = insieme di clausole = `testa :- letterale`$_1$`, ... letterale`$_n$`.`

```
criminal(X) :- american(X), weapon(Y), sells(X,Y,Z), hostile(Z).
```

Unificazione efficiente tramite open coding
Recupero efficiente di clausole "attivabili" per mezzo di direct linking
Depth-first, left-to-right backward chaining
Predicati built-in per l'aritmetica etc., e.g., `X is Y*Z+3`
Assunzione del mondo chiuso (Closed-world assumption) ("negazione come fallimento")
p.e., dato `vivo(X) :- not morto(X).`
`vivo(joe)` ha successo se `morto(joe)` fallisce

Esempi di Prolog

Ricerca depth-first da uno stato iniziale `X`:

```
dfs(X) :- goal(X).
dfs(X) :- successor(X,S),dfs(S).
```

Append di due liste:

```
append([],Y,Y).
append([X|L],Y,[X|Z]) :- append(L,Y,Z).
```

```
query:    append(A,B,[1,2]) ?
risposte: A=[]    B=[1,2]
          A=[1]   B=[2]
          A=[1,2] B=[]
```

Risoluzione: breve sommario

Versione completa del primo ordine:

$$\frac{\ell_1 \vee \cdots \vee \ell_k, \qquad m_1 \vee \cdots \vee m_n}{(\ell_1 \vee \cdots \vee \ell_{i-1} \vee \ell_{i+1} \vee \cdots \vee \ell_k \vee m_1 \vee \cdots \vee m_{j-1} \vee m_{j+1} \vee \cdots \vee m_n)\theta}$$

dove $\textsc{Unify}(\ell_i, \neg m_j) = \theta$.

Per esempio,

$$\frac{\begin{array}{l}\neg Rich(x) \vee Unhappy(x)\\ Rich(Ken)\end{array}}{Unhappy(Ken)}$$

con $\theta = \{x/Ken\}$

Applica passi di risoluzione a $CNF(KB \wedge \neg\alpha)$; completo per FOL

Conversione a CNF

Ognuno che ama tutti gli animali è amato da qualcuno:
$\forall x\ [\forall y\ \ (Animal(y)\ \Rightarrow\ Loves(x,y))]\ \Rightarrow\ [\exists y\ \ Loves(y,x)]$

1. Eliminare doppie e singole implicazioni

$$\forall x\ [\neg\forall y\ \ (\neg Animal(y) \vee Loves(x,y))] \vee [\exists y\ \ Loves(y,x)]$$

2. Spostare $\neg$ all'interno: $\neg\forall x, p\ \ \equiv \exists x\ \ \neg p$, $\ \neg\exists x, p\ \ \equiv \forall x\ \ \neg p$:

$$\forall x\ [\exists y\ \ \neg(\neg Animal(y) \vee Loves(x,y))] \vee [\exists y\ \ Loves(y,x)]$$
$$\forall x\ [\exists y\ \ \neg\neg Animal(y) \wedge \neg Loves(x,y)] \vee [\exists y\ \ Loves(y,x)]$$
$$\forall x\ [\exists y\ \ Animal(y) \wedge \neg Loves(x,y)] \vee [\exists y\ \ Loves(y,x)]$$

Conversione a CNF

3. Standardizzare variabili: ogni quantificatore deve usarne una differente

$$\forall x\ [\exists y\ \ Animal(y) \wedge \neg Loves(x,y)] \vee [\exists z\ \ Loves(z,x)]$$

4. Skolemizzare:
   ogni variabile esistenziale è rimpiazzata da una funzione di Skolem applicata a tutte quelle variabili quantificate universalmente nel cui scope compare il quantificatore esistenziale:

$$\forall x\ [Animal(F(x)) \wedge \neg Loves(x,F(x))] \vee Loves(G(x),x)$$

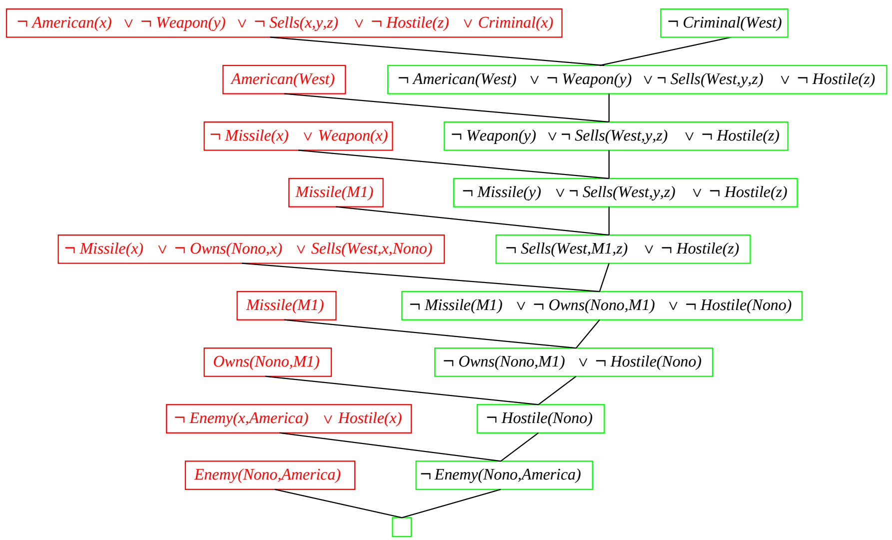
5. Rimuovere i quantificatori universali:

$$[Animal(F(x)) \wedge \neg Loves(x,F(x))] \vee Loves(G(x),x)$$

6. Distribuire $\wedge$ su $\vee$:

$$[Animal(F(x)) \vee Loves(G(x),x)] \wedge [\neg Loves(x,F(x)) \vee Loves(G(x),x)]$$

Prova di risoluzione per clausole definite



Strategie di risoluzione

◇ Unit Clause

Si preferisce effettuare la risoluzione con una delle sentenze costituita da un singolo letterale (clausola unitaria)

◇ Unit Resolution

Forma ristretta di risoluzione in cui ogni passo di risoluzione deve coinvolgere una clausola unitaria (incompleta in generale, completa per clausole di Horn)

◇ Set of Support

Basata sull'insieme di supporto da cui si preleva una delle sentenze e dove si pone il risolvente. Esempio di insieme di supporto: il goal (e tutti i risolventi derivati da esso)

◇ Input Resolution

Combina una sentenza in input (KB e Query) con il risolvente corrente. Tipica struttura a spina di pesce.

◇ Linear Resolution

Come Input Resolution, ma ammette anche la combinazione del risolvente corrente con suoi avi.

◇ Subsumption

Elimina tutte le sentenze che sono "subsumed" (più specifiche di altre). Es. $P(A)$ e $P(A) \vee P(B)$ sono più specifiche di $P(x)$.

Uguaglianza

Fino ad ora l'uguaglianza ($=$) non è stata trattata.

Esistono fondamentalmente 3 approcci diversi:

1. Assiomatizzazione
   Idea base: si aggiungono assiomi che descrivono le proprietà dell'uguaglianza ed assiomi opportuni per ogni predicato e funzione.

2. Aggiunta di una nuova regola di inferenza (p.e., Demodulation, Paramodulation)
   Idea base: se $x = y$ e $UNIFY(x, z) = \theta$,
   rimpiazza z con $SUBST(\theta, y)$.

3. Estensione dell'algoritmo di unificazione
   Idea base: unifica termini equivalenti,
   p.e. $1 + 2$ e $2 + 1$.

Riassunto

Inferenza in FOL

◇ Proposizionalizzazione ed uso di unificazione per evitare la istanziazione di variabili coinvolte in una prova.

◇ Modus Ponens Generalizzato (GMP) usa l'unificazione e permette l'applicazione di forward e backward chaining su clausole definite.

◇ GMP è completo per clausole definite, anche se determinare le conseguenze logiche è un problema semidecidibile. Per Datalog (no funzioni e clausole definite) il problema è decidibile.

◇ Forward chaining è completo e polinomiale per Datalog.

◇ Backward chaining è usato in Programmazione Logica (p.e., Prolog) e rinforzato con opportune tecniche di compilazione.

◇ Risoluzione generalizzata è completa per KB in forma normale congiuntiva (CNF). Se però si usa il principio di induzione $\Rightarrow$ non completa.