



Intelligenza Artificiale

Agenti Logici



Outline

- ▶ Agenti basati sulla conoscenza
- ▶ Il mondo del Wumpus
- ▶ Logica in generale
- ▶ Logica Proposizionale (Booleana)
- ▶ Equivalenza, validità, soddisfabilità
- ▶ Regole di inferenza
- ▶ Dimostrazione dei teoremi
 - ▶ resolution, forward/backward chaining
- ▶ Model Checking
 - ▶ DPLL, WalkSAT

Agenti basati sulla conoscenza

- ▶ Il componente più importante degli agenti basati sulla conoscenza è la

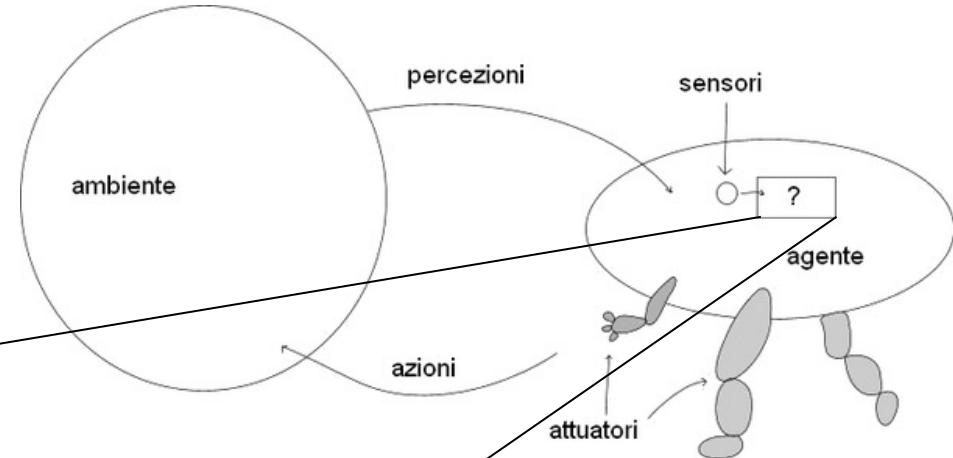
Base di conoscenza (KB, Knowledge Base)

- ▶ Un insieme di formule
 - ▶ espresse in un linguaggio di rappresentazione della conoscenza.
- ▶ Ogni formula rappresenta un'asserzione sul mondo
- ▶ Una formula è detta “assioma” quando non deve essere ricavata da altre formule

Agenti basati sulla conoscenza

- ▶ La base di conoscenza deve prevedere meccanismi per aggiungere nuove formule e per le interrogazioni
- ▶ Si usa un approccio dichiarativo attraverso due azioni standard:
 - ▶ **TELL** (Asserisci)
 - ▶ **ASK** (Chiedi)
- ▶ La risposta di ogni richiesta (**ASK**) posta alla base di conoscenza deve essere una conseguenza di quello che è stato detto (**TELL**) in precedenza.

Un semplice agente basato sulla conoscenza



```
function KB-AGENT(percept) returns an action
  persistent: KB, a knowledge base
  t, a counter, initially 0, indicating time
```

```
  TELL(KB, MAKE-PERCEPT-SENTENCE(percept, t))
  action  $\leftarrow$  ASK(KB, MAKE-ACTION-QUERY(t))
  TELL(KB, MAKE-ACTION-SENTENCE(action, t))
  t  $\leftarrow$  t + 1
  return action
```

Un semplice agente basato sulla conoscenza

```
function KB-AGENT(percept) returns an action
```

persistent: KB , a knowledge base
 t , a counter, initially 0, indicating time

```
    TELL( $KB$ , MAKE-PERCEPT-SENTENCE(percept,  $t$ ))
```

```
    action  $\leftarrow$  ASK( $KB$ , MAKE-ACTION-QUERY( $t$ ))
```

```
    TELL( $KB$ , MAKE-ACTION-SENTENCE(action,  $t$ ))
```

```
     $t \leftarrow t + 1$ 
```

```
return action
```

- ▶ Prende in input una percezione
- ▶ Restituisce un'azione

Un semplice agente basato sulla conoscenza

```
function KB-AGENT(percept) returns an action
  persistent: KB, a knowledge base
    t, a counter, initially 0, indicating time
```

```
  TELL(KB, MAKE-PERCEPT-SENTENCE(percept, t))
  action  $\leftarrow$  ASK(KB, MAKE-ACTION-QUERY(t))
  TELL(KB, MAKE-ACTION-SENTENCE(action, t))
  t  $\leftarrow$  t + 1
  return action
```

- ▶ Mantiene in memoria una KB che può contenere *conoscenza iniziale (background knowledge)*

Un semplice agente basato sulla conoscenza

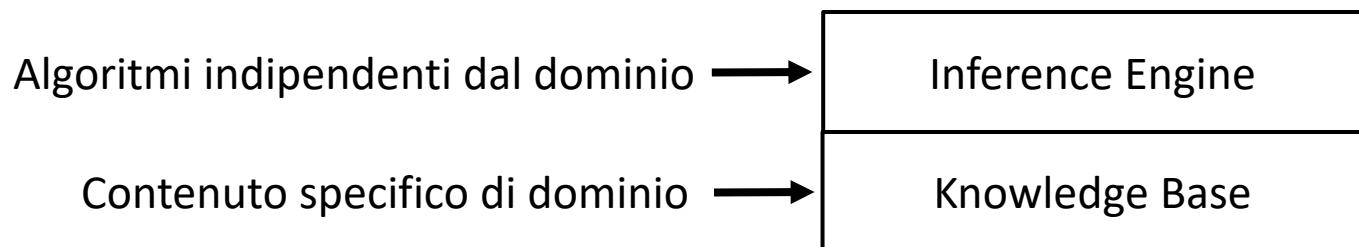
```
function KB-AGENT(percept) returns an action
    persistent: KB, a knowledge base
        t, a counter, initially 0, indicating time
```

```
    TELL(KB, MAKE-PERCEPT-SENTENCE(percept, t))
    action  $\leftarrow$  ASK(KB, MAKE-ACTION-QUERY(t))
    TELL(KB, MAKE-ACTION-SENTENCE(action, t))
    t  $\leftarrow$  t + 1
    return action
```

- ▶ Ogni volta che viene invocato il programma agente fa 3 cose:
 - ▶ Comunica le sue percezioni alla KB (**TELL**)
 - ▶ Chiede alla KB quale azione eseguire (**ASK**)
 - ▶ Registra l'azione scelta nella KB prima di esegirla (**TELL**)

Agenti basati sulla conoscenza

- ▶ Un agente può essere descritto sulla base di differenti livelli di astrazione
- ▶ Livello della conoscenza
 - ▶ cosa si conosce, a prescindere da come è implementato
- ▶ Livello dell'implementazione
 - ▶ le strutture di dati nella KB e gli algoritmi che li manipolano



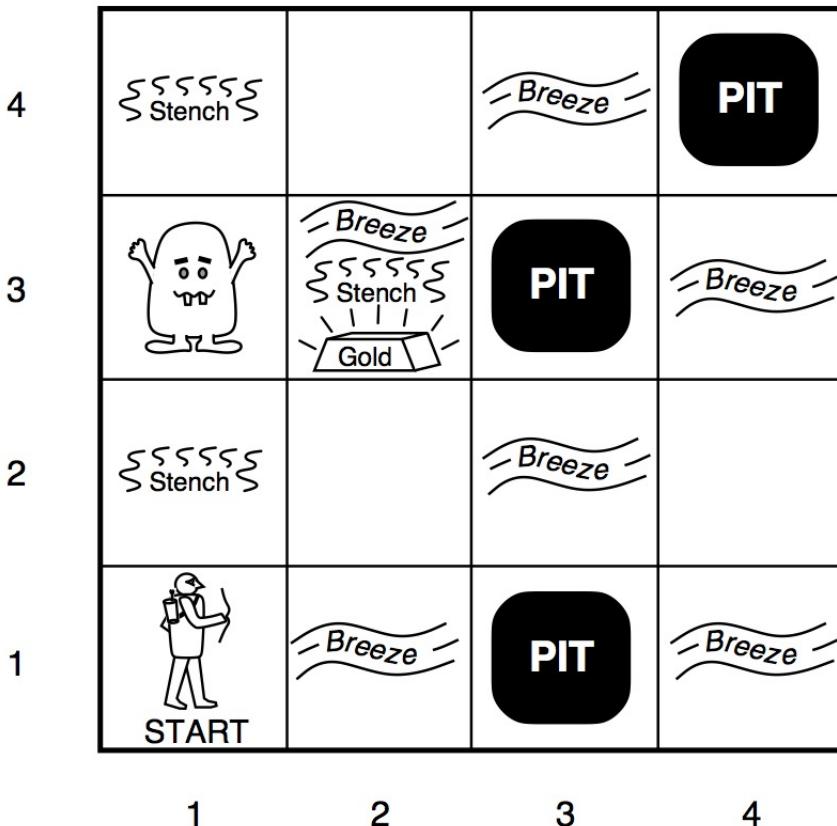
Outline

- ▶ Agenti basati sulla conoscenza
- ▶ Il mondo del Wumpus
- ▶ Logica in generale
- ▶ Logica Proposizionale (Booleana)
- ▶ Equivalenza, validità, soddisfabilità
- ▶ Regole di inferenza
- ▶ Dimostrazione dei teoremi
 - ▶ resolution, forward/backward chaining
- ▶ Model Checking
 - ▶ DPLL, WalkSAT

Il mondo del Wumpus

- ▶ Descriviamo un ambiente in cui gli agenti basati sulla conoscenza possono dimostrare il loro valore.
 - ▶ Una caverna formata da stanze collegate da passaggi.
 - ▶ L'agente vuole scovare l'oro.
 - ▶ **ATTENZIONE:** il wumpus e i pozzi nascosti nelle stanze sono pronti ad uccidere l'agente.
 - ▶ L'agente ha solo una freccia per uccidere il wumpus

4	Stench		Breeze
3		Breeze Stench Gold	
2	Stench		Breeze
1	START	Breeze	



Il mondo del Wumpus

- ▶ Misura di prestazioni
 - ▶ +1000 quando l'agente trova l'oro
 - ▶ -1000 quando l'agente muore
 - ▶ -1 per ogni azione eseguita
 - ▶ -10 per l'uso della freccia
- ▶ Descrizione dell'ambiente
 - ▶ Nelle celle adiacenti al wumpus c'è *Fetore (Stench)*
 - ▶ Nelle celle adiacenti ai pozzi c'è *Brezza (Breeze)*
 - ▶ Nella cella che contiene l'oro c'è *Scintillio (Glitter)*
 - ▶ L'agente ha solo una freccia per uccidere il wumpus
 - ▶ La freccia colpisce il wumpus solo se si trova nella direzione corretta
 - ▶ L'agente può prendere l'oro solo se si trova nella stessa cella

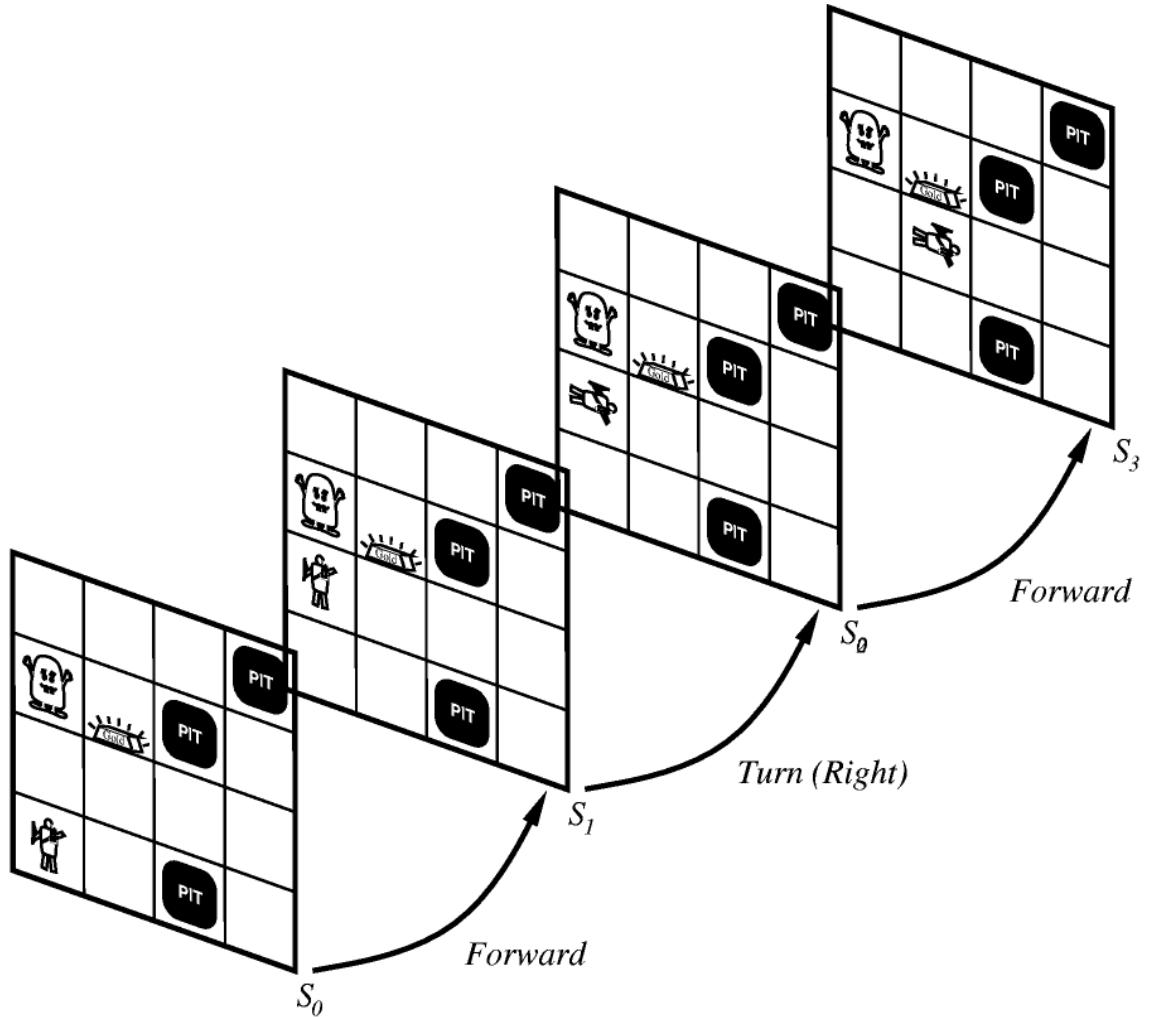
Il mondo del Wumpus

▶ Attuatori

- ▶ Girare a destra
- ▶ Girare a sinistra
- ▶ Andare avanti
- ▶ Afferrare
- ▶ Tirare
- ▶ Esci

▶ Sensori

- ▶ Fetore
- ▶ Brezza
- ▶ Scintillio
- ▶ Urto
- ▶ Urlo



Caratterizzazione del mondo del Wumpus

- ▶ ~~Completamente osservabile~~/Parzialmente osservabile
 - ▶ Alcuni aspetti dello stato non sono percepibili (status del wumpus)
- ▶ Agente singolo/~~Muliagente~~
 - ▶ Solo l'agente si può muovere
- ▶ Deterministico/~~Stocastico~~
 - ▶ Gli esisti vengono specificati esattamente
- ▶ ~~Episodico~~/Sequenziale
 - ▶ Si può raggiungere un premio soltanto dopo diverse azioni
- ▶ Statico/~~Dinamico~~
 - ▶ Il wumpus e i pozzi non si muovono
- ▶ Discreto/~~Continuo~~
 - ▶ Numero finito di stati distinti

Esplorare un mondo del Wumpus

A = Agente

B = Brezza

G = Scintillio (Glitter)

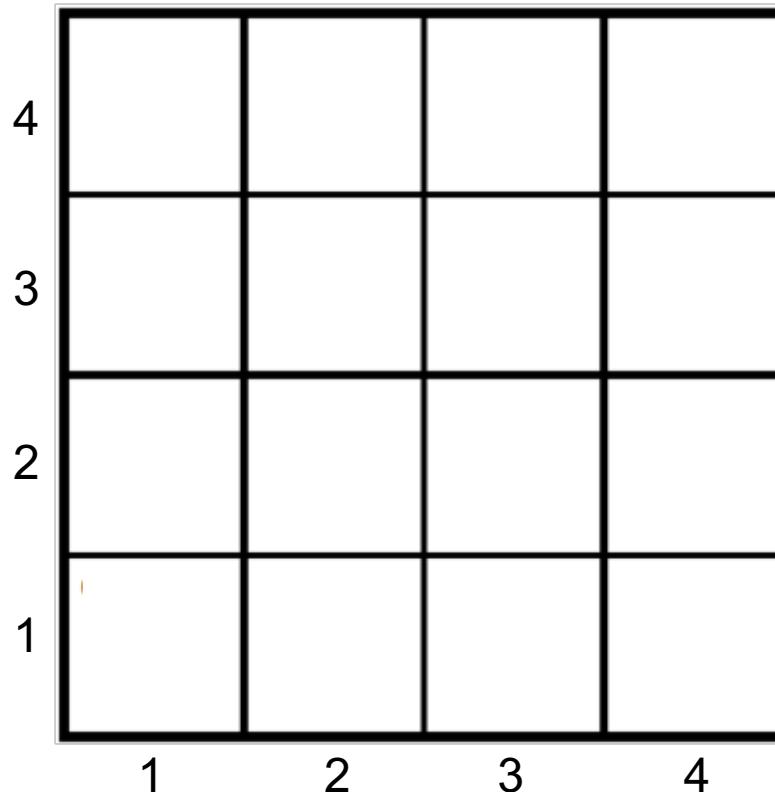
OK = Cella sicura

P = Pozzo

S = Fetore (Stench)

V = Cella visitata

W = Wumpus



Esplorare un mondo del Wumpus

▶ Cella [1,1]

La KB iniziale contiene le regole dell'ambiente

- ▶ La prima percezione è *[none,none,none,none,none]*
- ▶ Ci possiamo muovere verso una cella sicura, ad esempio la cella [1,2]

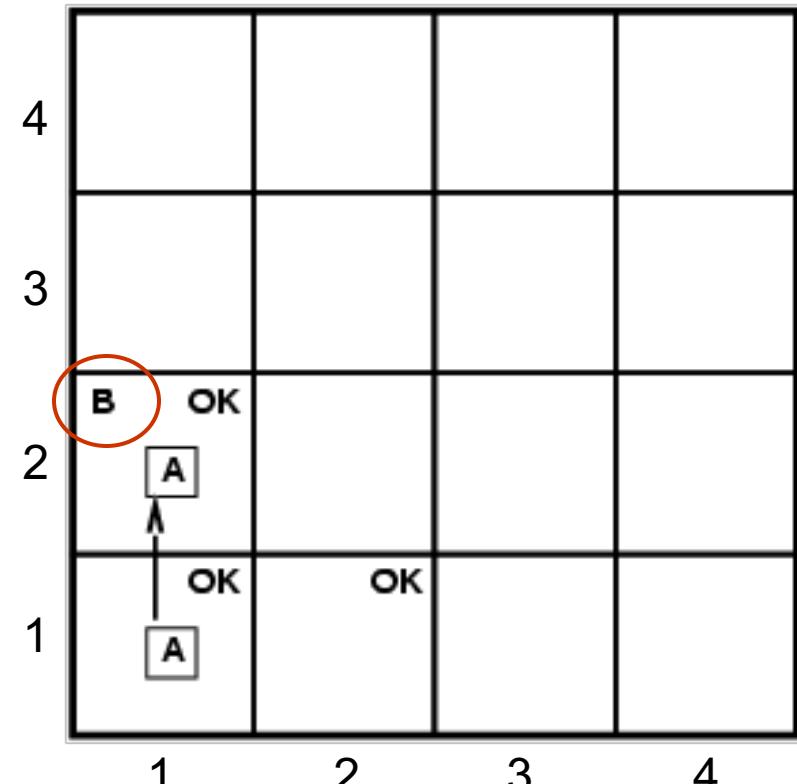
1,4	2,4	3,4	4,4
1,3	2,3	3,3	4,3
1,2	2,2	3,2	4,2
OK			
1,1 A OK	2,1 OK	3,1	4,1

Esplorare un mondo del Wumpus

▶ Cella [1,2]

La brezza indica che c'è un pozzo in [1,3] o in [2,2]

- ▶ La seconda percezione è *[none,brezza,none,none,none]*



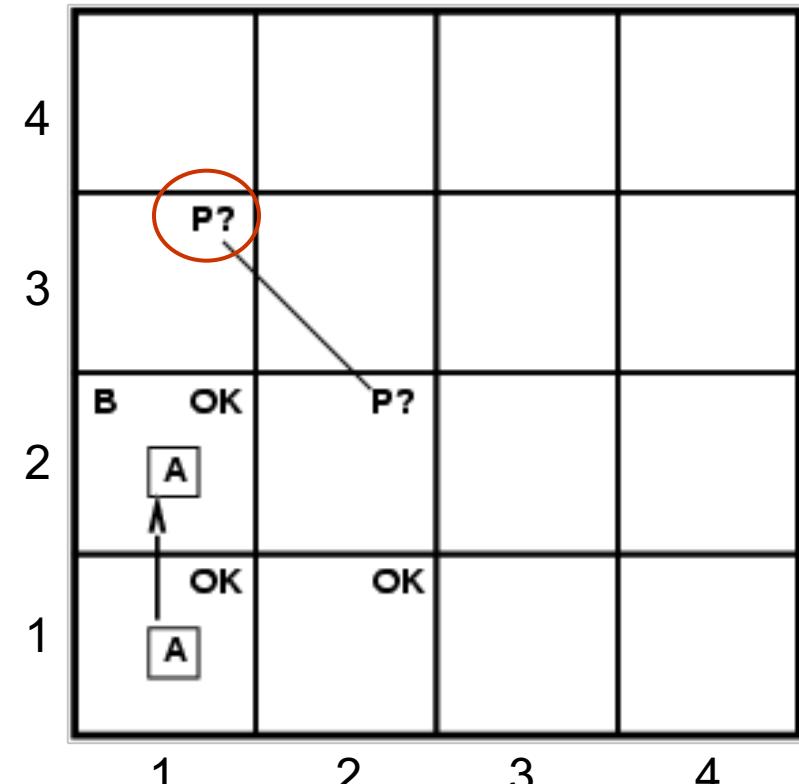
B = Brezza

Esplorare un mondo del Wumpus

▶ Cella [1,2]

La brezza indica che c'è un pozzo in [1,3] o in [2,2]

- ▶ La seconda percezione è *[none,brezza,none,none,none]*
- ▶ L'unica cella sicura è la [1,1]
- ▶ Dobbiamo tornare indietro e andare a destra (cella [2,1])



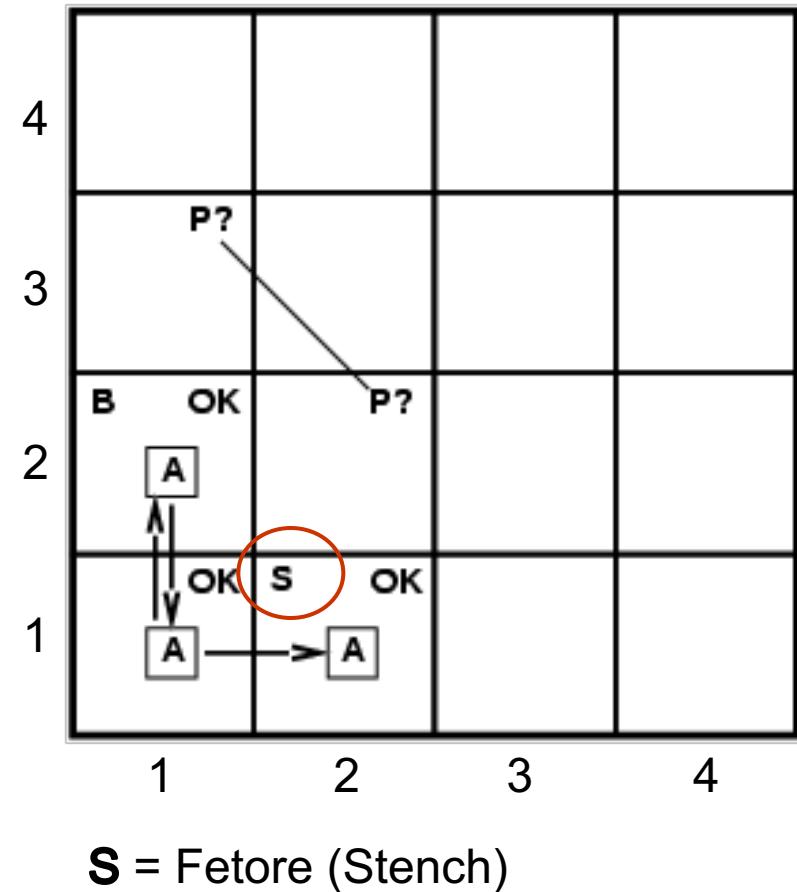
P = Pozzo

Esplorare un mondo del Wumpus

▶ Cella [2,1]

Il fetore indica che il wumpus è vicino

- ▶ La seconda percezione è *[fetore,none,none,none,none]*
- ▶ Il wumpus non può essere nella cella [1,1]
- ▶ Il wumpus non può essere nella cella [2,2], altrimenti l'agente avrebbe percepito puzza quando era in [1,2]

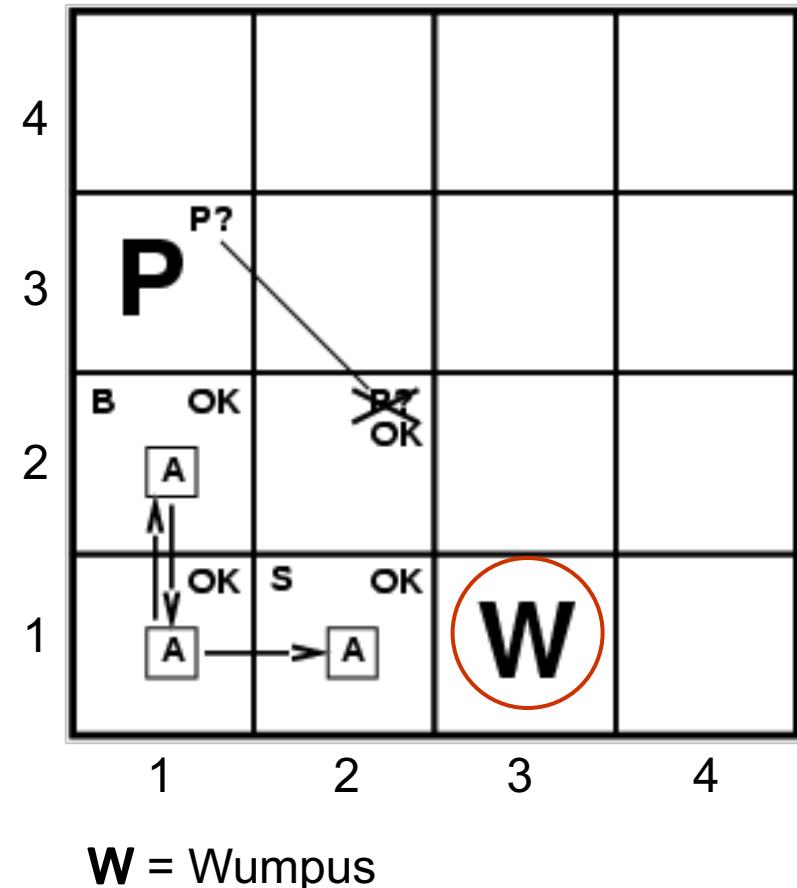


Esplorare un mondo del Wumpus

▶ Cella [2,1]

Il fetore indica che il wumpus è vicino

- ▶ La seconda percezione è *[fetore,none,none,none,none]*
- ▶ L'agente può dedurre che il wumpus si trova nella cella [3,1]
- ▶ La mancanza di brezza in [2,1] implica che non ci sono pozzi in [2,2]

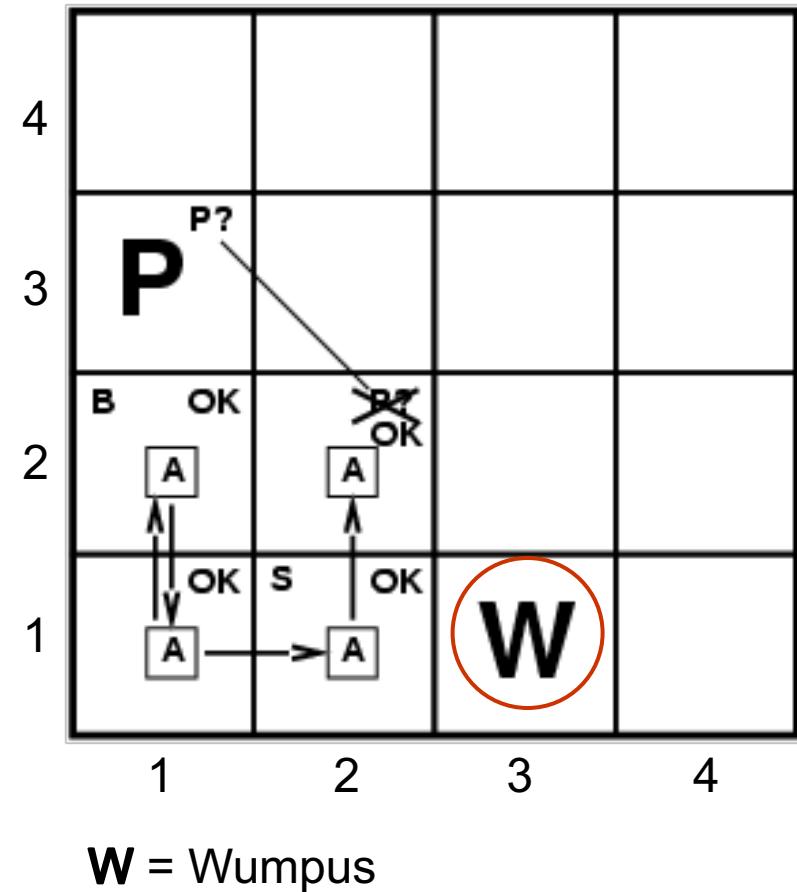


Esplorare un mondo del Wumpus

▶ Cella [2,2]

Non si percepisce nulla le celle vicine sono sicure

- ▶ La terza percezione è *[none,none,none,none,none]*
- ▶ Le celle vicine possono essere contrassegnate con OK

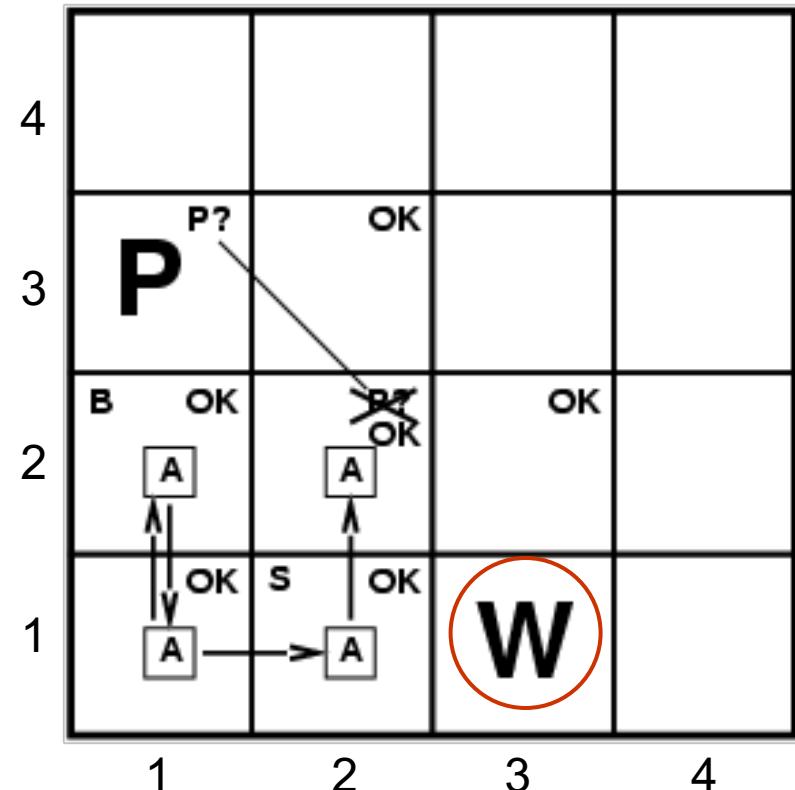


Esplorare un mondo del Wumpus

▶ Cella [2,2]

Non si percepisce nulla le celle vicine sono sicure

- ▶ La terza percezione è *[none,none,none,none,none]*
- ▶ Ci possiamo muovere verso una cella sicura, ad esempio la cella [3,2]



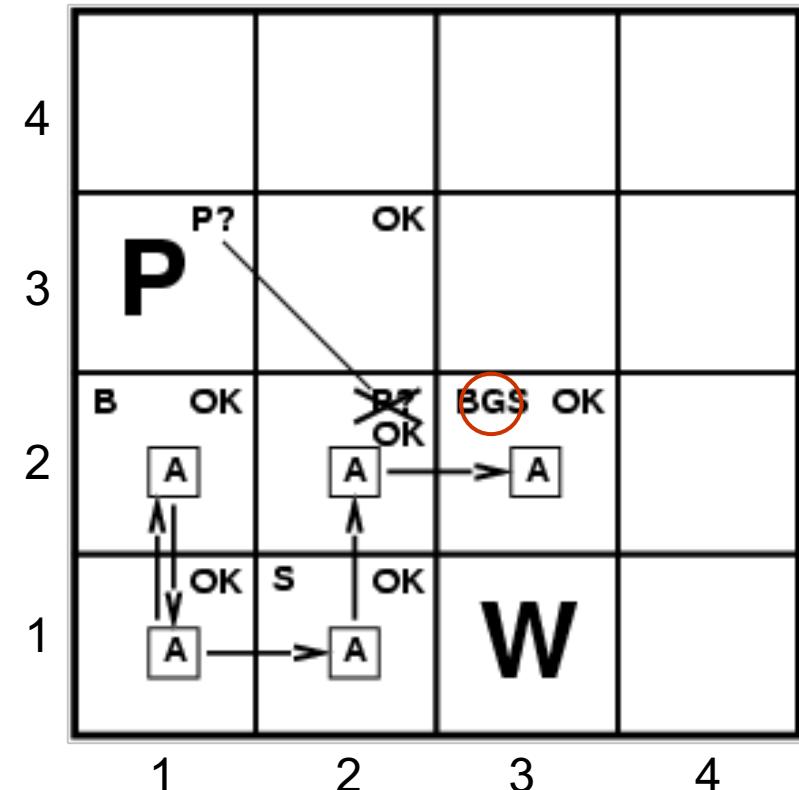
W = Wumpus

Esplorare un mondo del Wumpus

▶ Cella [3,2]

Lo scintillio indica che abbiamo trovato l'oro

- ▶ La terza percezione è *[fetore,brezza,scintillio,none,none]*
- ▶ L'agente può afferrare l'oro



G = Scintillio (Glitter)

Outline

- ▶ Agenti basati sulla conoscenza
- ▶ Il mondo del Wumpus
- ▶ Logica in generale
- ▶ Logica Proposizionale (Booleana)
- ▶ Equivalenza, validità, soddisfabilità
- ▶ Regole di inferenza
- ▶ Dimostrazione dei teoremi
 - ▶ resolution, forward/backward chaining
- ▶ Model Checking
 - ▶ DPLL, WalkSAT

Che cos'è la logica?

- ▶ Le logiche sono linguaggi formali per rappresentare le informazioni, in modo tale che possano essere tratte delle conclusioni
- ▶ Sintassi
 - ▶ che specifica come devono essere formattate le sentenze
- ▶ Semantica, che specifica il “significato” delle sentenze
 - ▶ La verità delle formule rispetto a ogni mondo possibile
- ▶ Esempio: il linguaggio dell’aritmetica
 - ▶ $x + 2 \geq y$ è una sentenza, $x2 + y >$ non è una sentenza
 - ▶ $x + 2 \geq y$ è vera in un mondo dove $x=7$ e $y=1$
 - ▶ $x + 2 \geq y$ è falsa in un mondo dove $x=0$ e $y=6$

Conseguenza logica

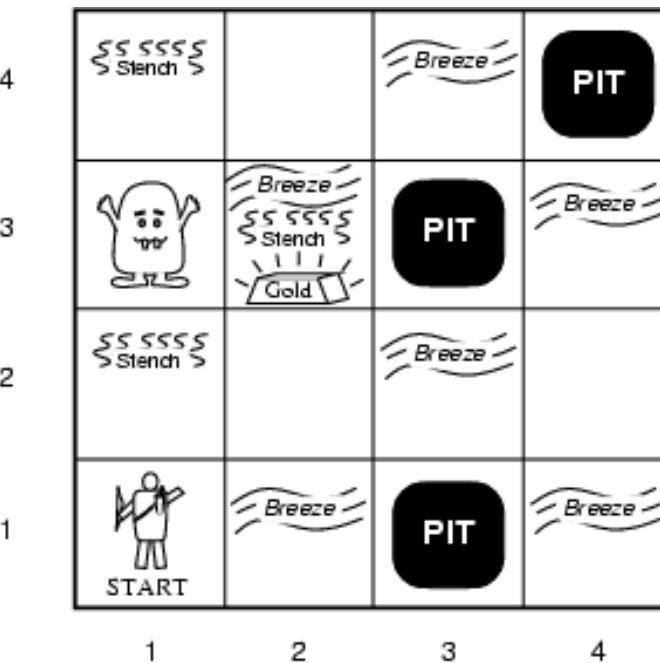
- ▶ Relazione di conseguenza logica tra formule
 - ▶ Una formula che *segue logicamente* da un'altra

$$KB \models \alpha$$

- ▶ KB implica la sentenza α se e soltanto se α è vera in tutti i mondi in cui KB è vera.
- ▶ Esempio: il linguaggio dell'aritmetica
 - ▶ $x + y = 4$ implica $4 = x + y$
- ▶ La conseguenza logica è una relazione tra sentenze (**sintassi**) che si basa sulla **semantica**.

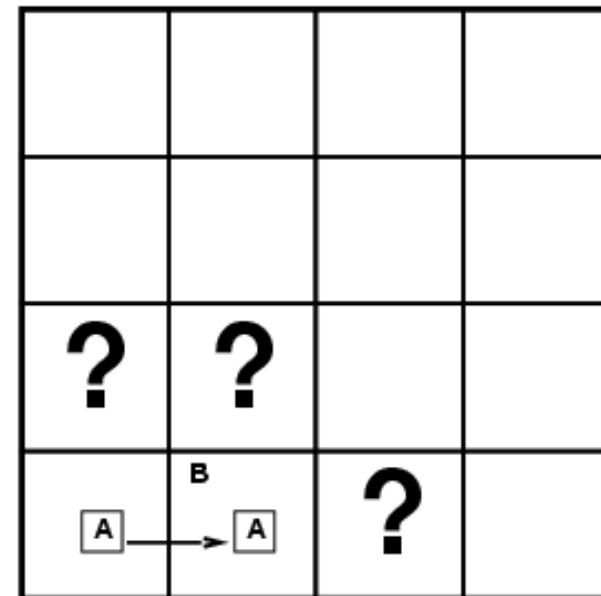
Conseguenza logica nel mondo del Wumpus

- ▶ Applichiamo l'analisi sulla conseguenza logica nell'esempio del mondo del wumpus

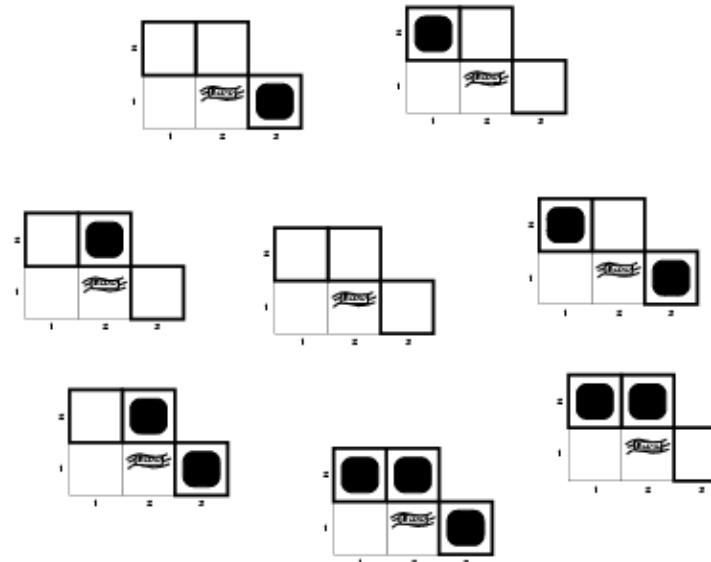


Conseguenza logica nel mondo del Wumpus

- ▶ Applichiamo l'analisi sulla conseguenza logica nell'esempio del mondo del wumpus
 - ▶ Consideriamo la seguente situazione
 - ▶ Nessuna percezione in [1,1]
 - ▶ Brezza in [2,1]
 - ▶ L'agente vuole sapere se le stanze adiacenti contengono pozzi
 - ▶ [1,2], [2,2], [3,1]
 - ▶ Il pozzo può esserci o meno (V/F)
 - ▶ $2^3 = 8$ possibili modelli

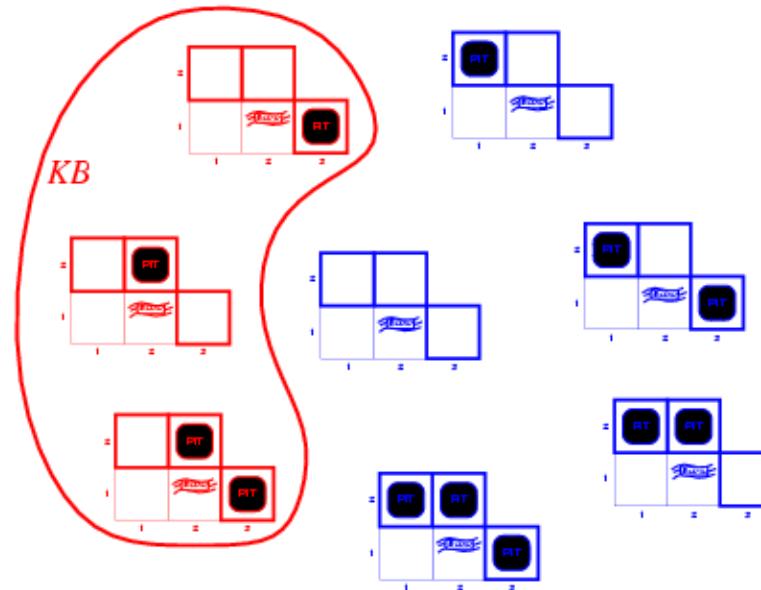


Modelli nel mondo del Wumpus



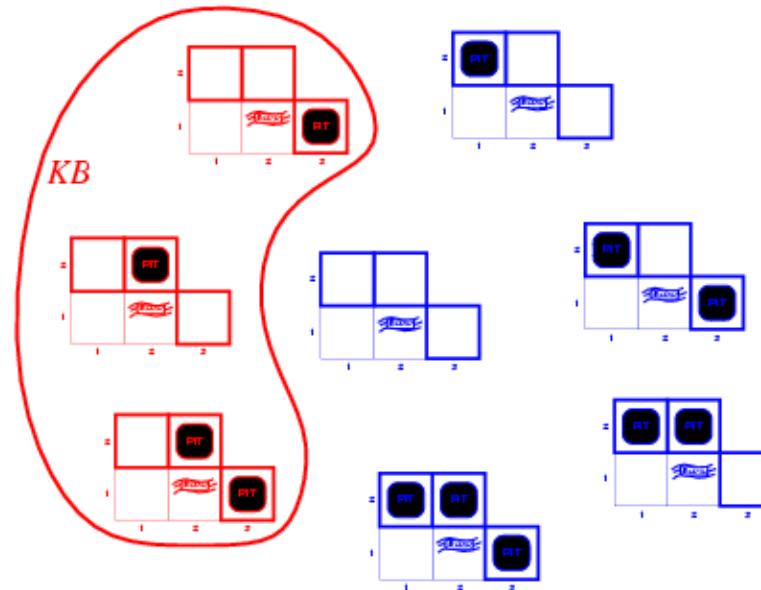
- ▶ La KB è falsa nei modelli che contraddicono le conoscenze dell'agente

Modelli nel mondo del Wumpus



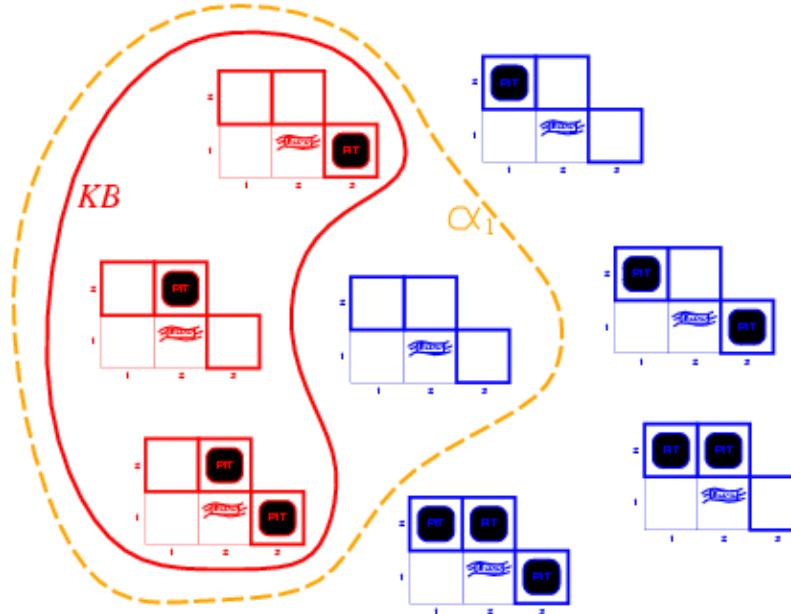
- ▶ I modelli in cui nella [1,2] c'è un pozzo sono falsi
- ▶ L'agente è stato nella cella [1,1] e non ha percepito brezza

Modelli nel mondo del Wumpus



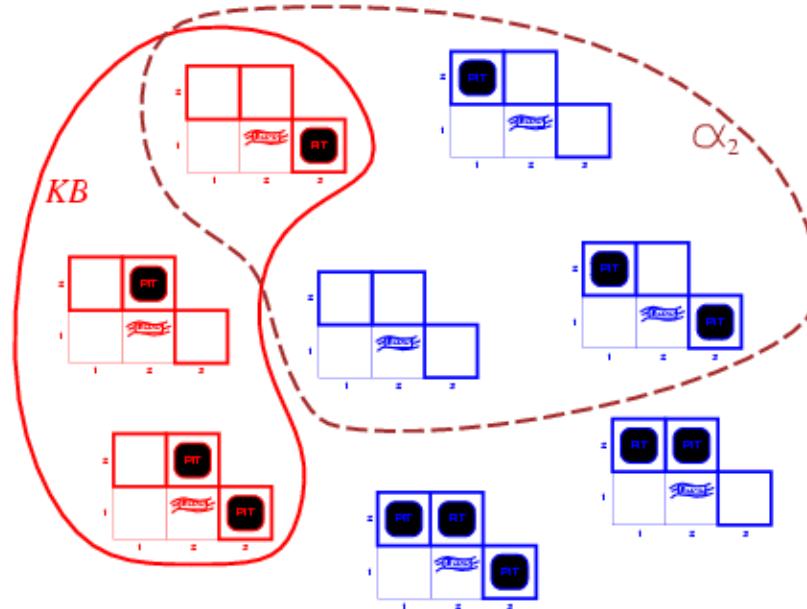
- ▶ Ora consideriamo le possibili conclusioni
 - ▶ α_1 = “Non c’è pozzo in [1,2]”
 - ▶ α_2 = “Non c’è pozzo in [2,2]”

Model checking



- ▶ α_1 = “Non c’è pozzo in [1,2]”
- ▶ In ogni modello in cui KB è vera, lo è anche α_1
- ▶ $KB \models \alpha_1$: Non c’è pozzo in [1,2]

Model checking



- ▶ $\alpha_2 = \text{"Non c'è pozzo in [2,2]"}$
- ▶ In alcuni modelli in cui KB è vera, α_2 non lo è
- ▶ $KB \not\models \alpha_2$: L'agente non può concludere che non ci sia pozzo in [2,2]

Conseguenza logica vs Inferenza

- ▶ Esempio:
 - ▶ KB è un pagliaio
 - ▶ α è un ago
- ▶ La conseguenza logica è come dire che l'ago si trova in un pagliaio
- ▶ L'inferenza equivale a trovarlo. Il processo precedente si chiama di **model checking**.

Inferenza

- ▶ α può essere *derivato* da KB attraverso un algoritmo (procedura) di inferenza i

$$KB \vdash_i \alpha$$

- ▶ **Correttezza (Soundness)**: i è corretto se ogni volta che vale $KB \vdash_i \alpha$, è anche vero che $KB \models \alpha$
 - ▶ Un algoritmo che deriva solo formule che sono conseguenze logiche
- ▶ **Completezza (Completeness)**: i è completo se ogni volta che vale $KB \models \alpha$, è anche vero che $KB \vdash_i \alpha$
 - ▶ Un algoritmo che deriva ogni formula che è conseguenza logica

Outline

- ▶ Agenti basati sulla conoscenza
- ▶ Il mondo del Wumpus
- ▶ Logica in generale
- ▶ Logica Proposizionale (Booleana)
- ▶ Equivalenza, validità, soddisfabilità
- ▶ Regole di inferenza
- ▶ Dimostrazione dei teoremi
 - ▶ resolution, forward/backward chaining
- ▶ Model Checking
 - ▶ DPLL, WalkSAT

Logica Proposizionale

- ▶ La logica proposizionale è una logica semplice
- ▶ Essa ci permette di illustrare le idee che sono alla base dei linguaggi logici

Logica Proposizionale: Sintassi

- ▶ La logica proposizionale è una logica semplice
- ▶ La **Sintassi** della logica proposizionale definisce le formule accettabili (sentenze).
 - ▶ I simboli proposizionali (S_1, S_2 , etc) sono **formule atomiche** che rappresentano una **proposizione** che può essere **vera** o **falsa**
 - ▶ È possibile costruire **formule complesse** grazie all'uso delle **parentesi e dei connettivi logici**
 - ▶ Se S è una sentenza, anche $\neg S$ è una sentenza (**negazione**)
 - ▶ Se S_1 e S_2 sono sentenze, anche $S_1 \wedge S_2$ è una sentenza (**congiunzione**)
 - ▶ Se S_1 e S_2 sono sentenze, anche $S_1 \vee S_2$ è una sentenza (**disgiunzione**)
 - ▶ Se S_1 e S_2 sono sentenze, anche $S_1 \Rightarrow S_2$ è una sentenza (**implicazione**)
 - ▶ Se S_1 e S_2 sono sentenze, anche $S_1 \Leftrightarrow S_2$ è una sentenza (**equivalenza/bicondizionale**)

Logica Proposizionale: Sintassi

- ▶ La logica proposizionale è una logica semplice
- ▶ La **Sintassi** della logica proposizionale definisce le formule accettabili (sentenze).
- ▶ Ecco una grammatica formale (BNF, Backus-Naur Form) della logica proposizionale

formula	\rightarrow	formulaAtomica formulaComplessa
formulaAtomica	\rightarrow	True False P Q R ...
formulaComplessa	\rightarrow	(formula) [formula]
	/	\neg formula
	/	(formula \wedge formula)
		(formula \vee formula)
		(formula \Rightarrow formula)
		(formula \Leftrightarrow formula)
PRECEDENZA OPERATORI	:	$\neg, \wedge, \vee, \Rightarrow, \Leftrightarrow$

Logica Proposizionale: Semantica

- ▶ La logica proposizionale è una logica semplice
- ▶ La **Semantica** della logica proposizionale definisce le regole usate per determinare il valore di verità di una formula nei confronti di un particolare modello.
- ▶ Ogni modello specifica per ogni simbolo se è vero o falso
 - ▶ Esempio:

$P_{1,2}$	$P_{2,2}$	$P_{3,1}$
False	True	False
- ▶ Con 3 simboli esistono $2^3 = 8$ possibili modelli

Logica Proposizionale: Semantica

- ▶ La logica proposizionale è una logica semplice
- ▶ La **Semantica** della logica proposizionale deve specificare, dato un modello, come si fa a calcolare il valore di verità di qualsiasi formula
- ▶ Le regole per valutare il valore di verità rispetto ad un modello m sono le seguenti:

$\neg S$	è vero sse	S è falso		
$S_1 \wedge S_2$	è vero sse	S_1 è vero	and	S_2 è vero
$S_1 \vee S_2$	è vero sse	S_1 è vero	or	S_2 è vero
$S_1 \Rightarrow S_2$	è vero sse	S_1 è falso	or	S_2 è vero
i.e.,	è falso sse	S_1 è vero	and	S_2 è falso
$S_1 \Leftrightarrow S_2$	è vero sse	$S_1 \Rightarrow S_2$ è vero	and	$S_2 \Rightarrow S_1$ è vero

Logica Proposizionale: Semantica

- ▶ La logica proposizionale è una logica semplice
- ▶ La **Semantica** della logica proposizionale deve specificare, dato un modello, come si fa a calcolare il valore di verità di qualsiasi formula
- ▶ Le regole per valutare il valore di verità possono essere espresse attraverso **tavole di verità**

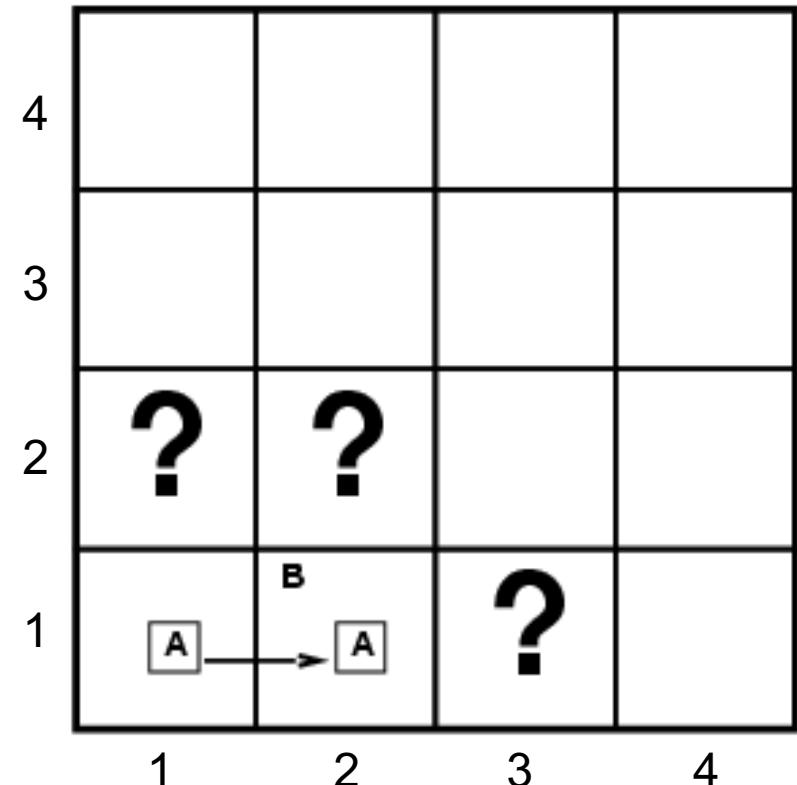
P	Q	$\neg P$	$P \wedge Q$	$P \vee Q$	$P \Rightarrow Q$	$P \Leftrightarrow Q$
<i>false</i>	<i>false</i>	<i>true</i>	<i>false</i>	<i>false</i>	<i>true</i>	<i>true</i>
<i>false</i>	<i>true</i>	<i>true</i>	<i>false</i>	<i>true</i>	<i>true</i>	<i>false</i>
<i>true</i>	<i>false</i>	<i>false</i>	<i>false</i>	<i>true</i>	<i>false</i>	<i>false</i>
<i>true</i>	<i>true</i>	<i>false</i>	<i>true</i>	<i>true</i>	<i>true</i>	<i>true</i>

Sentenze per il mondo del Wumpus

- ▶ Sulla base della logica proposizionale possiamo costruire una base di conoscenza per il mondo del wumpus.
- ▶ Concentriamoci sugli aspetti “immutabili” di tale mondo
- ▶ Per ora, ci servono i seguenti simboli per ogni posizione $[x,y]$
 - ▶ $P_{x,y}$ è vero se esiste un pozzo in $[x,y]$
 - ▶ $W_{x,y}$ è vero se esiste un wumpus in $[x,y]$, morto o vivo
 - ▶ $B_{x,y}$ è vero se l’agente percepisce una brezza in $[x,y]$
 - ▶ $S_{x,y}$ è vero se l’agente percepisce un fetore in $[x,y]$

Sentenze per il mondo del Wumpus

- ▶ In [1,1] non ci sono pozzi né brezza
 - ▶ $R_1: \neg P_{1,1}$
 - ▶ $R_4: \neg B_{1,1}$
- ▶ In [2,1] c'è brezza
 - ▶ $R_5: B_{2,1}$
- ▶ In una cella si percepisce brezza sse c'è un pozzo in una cella adiacente
 - ▶ $R_2: B_{1,1} \Leftrightarrow (P_{1,2} \vee P_{2,1})$
 - ▶ $R_3: B_{2,1} \Leftrightarrow (P_{1,1} \vee P_{2,2} \vee P_{3,1})$



Inferenza per il mondo del Wumpus

- ▶ Per costruire un primo algoritmo di inferenza utilizziamo l'approccio model checking
 - ▶ Un'implementazione diretta della definizione di conseguenza logica
- ▶ Data una sentenza α (ad esempio $\neg P_{1,2}$), enumeriamo esplicitamente i modelli e verifichiamo se α è vera in ogni modello in cui KB lo è
 - ▶ 7 simboli proposizionali
 - ▶ $2^7 = 128$ modelli

Inferenza per il mondo del Wumpus

- ▶ Data una sentenza α , enumeriamo esplicitamente i modelli e verifichiamo se α è vera in ogni modello in cui KB lo è

$B_{1,1}$	$B_{2,1}$	$P_{1,1}$	$P_{1,2}$	$P_{2,1}$	$P_{2,2}$	$P_{3,1}$	R_1	R_2	R_3	R_4	R_5	KB
false	true	true	true	true	false	false						
false	false	false	false	false	false	true	true	true	false	true	false	false
:	:	:	:	:	:	:	:	:	:	:	:	:
false	true	false	false	false	false	false	true	true	false	true	true	false
false	true	false	false	false	false	true	true	true	true	true	true	true
false	true	false	false	false	false	true	true	true	true	true	true	true
false	true	false	false	true	false	false	true	false	false	true	true	false
:	:	:	:	:	:	:	:	:	:	:	:	:
true	false	true	true	false	true	false						

Un algoritmo per calcolare conseguenze logiche

- ▶ Un algoritmo per calcolare le conseguenze logiche nella logica proposizionale

```
fuction TV-CONSEGU?(KB, α) returns true oppure false
    inputs: KB, la base di conoscenza, una formula logica proposizionale
            α, la query, una formula logica proposizionale

    simboli ← una lista di simboli proposizionali contenuti in KB e α
    return TV-VERIFICA-TUTTO(KB, α, simboli, {})

fuction TV-VERIFICA-TUTTO(KB, α, simboli, modello) returns true oppure false
    if VUOTO?(simboli) then
        if CP-VERO?(KB, modello) then return CP-VERO?(α, modello)
        else return true //quando KB è false, restituisce sempre true
    else do
        P ← PRIMO(simboli); resto ← RESTO(simboli);
        return TV-VERIFICA-TUTTO(KB, α, resto, modello ∪ {P = true}) and
               TV-VERIFICA-TUTTO(KB, α, resto, modello ∪ {P = false})
```

- ▶ Esegue l'enumerazione ricorsiva
 - ▶ È corretto e completo

Un algoritmo per calcolare conseguenze logiche

- ▶ Un algoritmo per calcolare le conseguenze logiche nella logica proposizionale

```
fuction TV-CONSEGU?(KB, α) returns true oppure false
    inputs: KB, la base di conoscenza, una formula logica proposizionale
            α, la query, una formula logica proposizionale

    simboli ← una lista di simboli proposizionali contenuti in KB e α
    return TV-VERIFICA-TUTTO(KB, α, simboli, {})

fuction TV-VERIFICA-TUTTO(KB, α, simboli, modello) returns true oppure false
    if VUOTO?(simboli) then
        if CP-VERO?(KB, modello) then return CP-VERO?(α, modello)
        else return true //quando KB è false, restituisce sempre true
    else do
        P ← PRIMO(simboli); resto ← RESTO(simboli);
        return TV-VERIFICA-TUTTO(KB, α, resto, modello ∪ {P = true}) and
               TV-VERIFICA-TUTTO(KB, α, resto, modello ∪ {P = false})
```

- ▶ Per n simboli la complessità temporale è $O(2^n)$

Outline

- ▶ Agenti basati sulla conoscenza
- ▶ Il mondo del Wumpus
- ▶ Logica in generale
- ▶ Logica Proposizionale (Booleana)
- ▶ Equivalenza, validità, soddisfabilità
- ▶ Regole di inferenza
- ▶ Dimostrazione dei teoremi
 - ▶ resolution, forward/backward chaining
- ▶ Model Checking
 - ▶ DPLL, WalkSAT

Equivalenza Logica

- ▶ Due formule α e β sono **logicamente equivalenti** se sono vere nello stesso insieme di modelli ($\alpha \equiv \beta$).
- ▶ Ecco alcune equivalenze logiche standard

$(\alpha \wedge \beta)$	\equiv	$(\beta \wedge \alpha)$ commutatività di \wedge
$(\alpha \vee \beta)$	\equiv	$(\beta \vee \alpha)$ commutatività di \vee
$((\alpha \wedge \beta) \wedge \gamma)$	\equiv	$(\alpha \wedge (\beta \wedge \gamma))$ associatività di \wedge
$((\alpha \vee \beta) \vee \gamma)$	\equiv	$(\alpha \vee (\beta \vee \gamma))$ associatività di \vee
$\neg(\neg \alpha)$	\equiv	α eliminazione della doppia negazione
$(\alpha \Rightarrow \beta)$	\equiv	$(\neg \alpha \Rightarrow \neg \beta)$ contrapposizione
$(\alpha \Rightarrow \beta)$	\equiv	$(\neg \alpha \vee \beta)$ eliminazione dell'implicazione
$(\alpha \Leftrightarrow \beta)$	\equiv	$((\alpha \Rightarrow \beta) \wedge (\beta \Rightarrow \alpha))$ eliminazione del bicondizionale
$\neg(\alpha \wedge \beta)$	\equiv	$(\neg \alpha \vee \neg \beta)$ De Morgan
$\neg(\alpha \vee \beta)$	\equiv	$(\neg \alpha \wedge \neg \beta)$ De Morgan
$(\alpha \wedge (\beta \vee \gamma))$	\equiv	$((\alpha \wedge \beta) \vee (\alpha \wedge \gamma))$ distributività di \wedge su \vee
$(\alpha \vee (\beta \wedge \gamma))$	\equiv	$((\alpha \vee \beta) \wedge (\alpha \vee \gamma))$ distributività di \vee su \wedge

Equivalenza Logica

- ▶ Definizione alternativa
 - ▶ Due formule α e β sono equivalenti soltanto se ognuna di esse è conseguenza logica dell'altra

$$\alpha \equiv \beta \text{ se e solo se } \alpha \models \beta \text{ e } \beta \models \alpha$$

Validità

- ▶ Una formula è **valida** se è vera in *tutti* i modelli.
- ▶ $A \vee \neg A$ è sempre **valida**
- ▶ Le formule valide sono note come **tautologie** e sono *necessariamente* vere.

- ▶ La validità è connessa all'inferenza attraverso il **Teorema di Deduzione**:
 - ▶ $KB \models \alpha$ se e soltanto se $(KB \Rightarrow \alpha)$ è valida
 - ▶ Tale teorema afferma che ogni formula di implicazione valida descrive un'inferenza legittima.

Soddisfacibilità

- ▶ Una formula è **soddisfacibile** se è vera in, o soddisfatta da, *qualche* modello.
 - ▶ $R_1 \wedge R_2 \wedge R_3 \wedge R_4 \wedge R_5$ vera in 3 modelli
 - ▶ SAT: problema di verificare la soddisfacibilità di formule della logica proposizionale (NP-completo)
- ▶ Una formula è **insoddisfacibile** se è vera in *nessun* modello.
 - ▶ $A \wedge \neg A$
- ▶ La soddisfacibilità è connessa all'inferenza attraverso:
 - ▶ $\text{KB} \models \alpha$ se e soltanto se $(\text{KB} \wedge \neg \alpha)$ è insoddisfacibile

Outline

- ▶ Agenti basati sulla conoscenza
- ▶ Il mondo del Wumpus
- ▶ Logica in generale
- ▶ Logica Proposizionale (Booleana)
- ▶ Equivalenza, validità, soddisfabilità
- ▶ Regole di inferenza
- ▶ Dimostrazione dei teoremi
 - ▶ resolution, forward/backward chaining
- ▶ Model Checking
 - ▶ DPLL, WalkSAT

Regole di Inferenza

- ▶ Le regole di inferenza possono essere usate per derivare una dimostrazione o prova
 - ▶ Una catena di conclusioni che portano all'obiettivo desiderato
-
- ▶ Esistono diversi tipi di regole di inferenza:
 - ▶ Modus Ponens
$$\frac{\alpha \Rightarrow \beta, \quad \alpha}{\beta}$$
 - ▶ Ogni volta che sono date le formule $\alpha \Rightarrow \beta$ e α , allora si può inferire la formula β

Regole di Inferenza

- ▶ Le regole di inferenza possono essere usate per derivare una dimostrazione o prova
 - ▶ Una catena di conclusioni che portano all'obiettivo desiderato
-
- ▶ Esistono diversi tipi di regole di inferenza:
 - ▶ Modus Ponens
- $$\frac{(WumpusDavanti \wedge WumpusVivo) \Rightarrow ScagliaFreccia, \quad (WumpusDavanti \wedge WumpusVivo)}{ScagliaFreccia}$$
-
- ▶ Se è data $(WumpusDavanti \wedge WumpusVivo) \Rightarrow ScagliaFreccia$ e vale $(WumpusDavanti \wedge WumpusVivo)$, allora si può inferire $ScagliaFreccia$

Regole di Inferenza

- ▶ Le regole di inferenza possono essere usate per derivare una dimostrazione o prova
 - ▶ Una catena di conclusioni che portano all'obiettivo desiderato
-
- ▶ Esistono diversi tipi di regole di inferenza:
 - ▶ Eliminazione degli AND
$$\frac{\alpha \wedge \beta}{\alpha}$$
 - ▶ Data una congiunzione, si può inferire uno qualsiasi dei congiunti

Regole di Inferenza

- ▶ Le regole di inferenza possono essere usate per derivare una dimostrazione o prova
 - ▶ Una catena di conclusioni che portano all'obiettivo desiderato
-
- ▶ Esistono diversi tipi di regole di inferenza:
 - ▶ Eliminazione degli AND
$$\frac{WumpusDavanti \wedge WumpusVivo}{WumpusVivo}$$
 - ▶ Se è data $(WumpusDavanti \wedge WumpusVivo)$, allora si può inferire $WumpusVivo$

Regole di Inferenza

- ▶ Esistono diversi tipi di regole di inferenza:
 - ▶ Equivalenze logiche

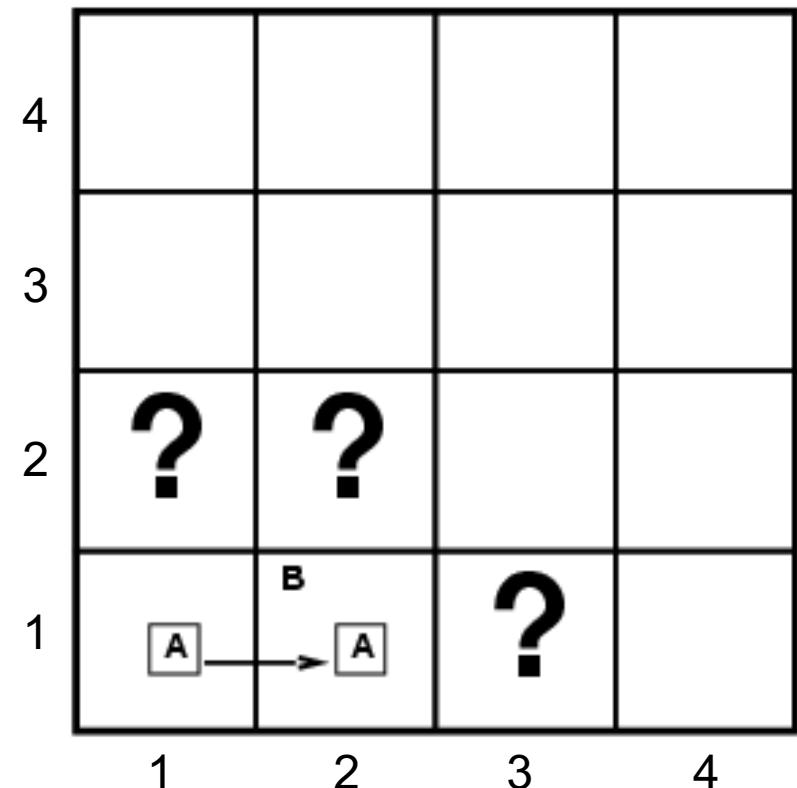
$(\alpha \wedge \beta)$	\equiv	$(\beta \wedge \alpha)$ commutatività di \wedge
$(\alpha \vee \beta)$	\equiv	$(\beta \vee \alpha)$ commutatività di \vee
$((\alpha \wedge \beta) \wedge \gamma)$	\equiv	$(\alpha \wedge (\beta \wedge \gamma))$ associatività di \wedge
$((\alpha \vee \beta) \vee \gamma)$	\equiv	$(\alpha \vee (\beta \vee \gamma))$ associatività di \vee
$\neg(\neg \alpha)$	\equiv	α eliminazione della doppia negazione
$(\alpha \Rightarrow \beta)$	\equiv	$(\neg \beta \Rightarrow \neg \alpha)$ contrapposizione
$(\alpha \Rightarrow \beta)$	\equiv	$(\neg \alpha \vee \beta)$ eliminazione dell'implicazione
$(\alpha \Leftrightarrow \beta)$	\equiv	$((\alpha \Rightarrow \beta) \wedge (\beta \Rightarrow \alpha))$ eliminazione del bicondizionale
$\neg(\alpha \wedge \beta)$	\equiv	$(\neg \alpha \vee \neg \beta)$ De Morgan
$\neg(\alpha \vee \beta)$	\equiv	$(\neg \alpha \wedge \neg \beta)$ De Morgan
$(\alpha \wedge (\beta \vee \gamma))$	\equiv	$((\alpha \wedge \beta) \vee (\alpha \wedge \gamma))$ distributività di \wedge su \vee
$(\alpha \vee (\beta \wedge \gamma))$	\equiv	$((\alpha \vee \beta) \wedge (\alpha \vee \gamma))$ distributività di \vee su \wedge

- ▶ Esempio

$$\frac{\alpha \Leftrightarrow \beta}{(\alpha \Rightarrow \beta) \wedge (\beta \Rightarrow \alpha)}$$

Esempio di trasformazione

- ▶ In [1,1] non ci sono pozzi né brezza
 - ▶ $R_1: \neg P_{1,1}$
 - ▶ $R_4: \neg B_{1,1}$
- ▶ In [2,1] c'è brezza
 - ▶ $R_5: B_{2,1}$
- ▶ In una cella si percepisce brezza sse c'è un pozzo in una cella adiacente
 - ▶ $R_2: B_{1,1} \Leftrightarrow (P_{1,2} \vee P_{2,1})$
 - ▶ $R_3: B_{2,1} \Leftrightarrow (P_{1,1} \vee P_{2,2} \vee P_{3,1})$



Esempio di trasformazione

- ▶ Vogliamo dimostrare $\neg P_{1,2}$
- ▶ Applichiamo la regola di De Morgan su R_9

1. $R_2: B_{1,1} \Leftrightarrow (P_{1,2} \vee P_{2,1})$
2. $R_6: (B_{1,1} \Rightarrow (P_{1,2} \vee P_{2,1})) \wedge ((P_{1,2} \vee P_{2,1}) \Rightarrow B_{1,1})$
3. $R_7: ((P_{1,2} \vee P_{2,1}) \Rightarrow B_{1,1})$
4. $R_8: (\neg B_{1,1} \Rightarrow \neg(P_{1,2} \vee P_{2,1}))$
5. $R_9: \neg(P_{1,2} \vee P_{2,1})$
6. $R_{10}: \neg P_{1,2} \wedge \neg P_{2,1}$

Algoritmi per dimostrazioni

- ▶ Possiamo applicare uno degli algoritmi di ricerca non informata per trovare una sequenza di passi che costituisca la dimostrazione
- ▶ **STATO INIZIALE:** la base di conoscenza (KB)
- ▶ **AZIONI:** l'insieme delle azioni consiste di tutte le regole di inferenza applicate a tutte le formule che corrispondono alla metà superiore della regola di inferenza
- ▶ **RISULTATO:** il risultato di un'azione è l'aggiunta della formula nella metà inferiore della regola di inferenza
- ▶ **OBIETTIVO:** arrivare ad uno stato che contiene la formula che stiamo cercando di dimostrare

Outline

- ▶ Agenti basati sulla conoscenza
- ▶ Il mondo del Wumpus
- ▶ Logica in generale
- ▶ Logica Proposizionale (Booleana)
- ▶ Equivalenza, validità, soddisfabilità
- ▶ Regole di inferenza
- ▶ Dimostrazione dei teoremi
 - ▶ resolution, forward/backward chaining
- ▶ Model Checking
 - ▶ DPLL, WalkSAT

Dimostrazione dei teoremi

- ▶ Le metodologie per effettuare la dimostrazione dei teoremi si dividono in due categorie principali
 - ▶ **Regole di Inferenza**
 - ▶ Generazione di nuove sentenze a partire dalle precedenti
 - ▶ **Dimostrazione** = un'applicazione sequenziale di regole di inferenza può usare queste ultime come operatori di un algoritmo di ricerca standard
 - ▶ Di solito è richiesta la trasformazione delle sentenze in qualche **forma normale**
 - ▶ **Model checking**
 - ▶ Enumerazione della tavola di verità (sempre esponenziale in n)
 - ▶ Backtracking migliorato, attraverso l'algoritmo *Davis-Putnam-Logemann-Loveland (DPLL)*
 - ▶ Ricerca euristica nello spazio dei modelli (corretto ma non completo), attraverso gli algoritmi *min-conflicts-like hill-climbing*

Dimostrazione per risoluzione

- ▶ Le regole di inferenza viste finora sono corrette, ma non abbiamo discusso della completezza degli algoritmi di inferenza
- ▶ Un’ulteriore regola d’inferenza: **la risoluzione**
 - ▶ Questa regola unita a qualsiasi algoritmo di ricerca completo dà luogo ad un algoritmo di inferenza completo
- ▶ Risoluzione unitaria

$$\frac{l_1 \vee \dots \vee l_k, \quad m}{l_1 \vee \dots \vee l_{i-1} \vee l_{i+1} \vee \dots \vee l_k}$$

con **m** complemento di l_i

- ▶ dove ogni **l** è un letterale e l_i ed **m** sono letterali complementari (cioè uno è la negazione dell’altro)
- ▶ La regola di risoluzione unitaria, prende una clausola (una disgiunzione di letterali) e un letterale e produce una nuova clausola

Dimostrazione per risoluzione

- ▶ Un’ulteriore regola d’inferenza: la **risoluzione**
 - ▶ Questa regola unita a qualsiasi algoritmo di ricerca completo dà luogo ad un algoritmo di inferenza completo
- ▶ Risoluzione completa

$$\frac{l_1 \vee \dots \vee l_k, \quad m_1 \vee \dots \vee m_n}{l_1 \vee \dots \vee l_{i-1} \vee l_{i+1} \vee \dots \vee l_k \vee m_1 \vee \dots \vee m_{j-1} \vee m_{j+1} \vee \dots \vee m_n}$$

- ▶ dove l_i ed m_j sono letterali complementari (cioè uno è la negazione dell’altro)
- ▶ La risoluzione prende due clausole e ne produce una nuova che contiene tutti i letterali delle due clausole originali tranne i due complementari
 - ▶ Esempio con clausole composte da due letterali:

$$\frac{l_1 \vee l_2, \quad \neg l_2 \vee l_3}{l_1 \vee l_3}$$

Esempio di risoluzione

- ▶ Supponiamo che l'agente torni da [2,1] a [1,1] e da lì passi in [1,2] dove percepisce una forte puzza, ma nessun movimento d'aria
 - ▶ $R_{11}: \neg B_{1,2}$
 - ▶ $R_{12}: B_{1,2} \Leftrightarrow (P_{1,1} \vee P_{2,2} \vee P_{1,3})$
- ▶ Applichiamo lo stesso processo che ci ha portato ad R_{10} ed otteniamo
 - ▶ $R_{13}: \neg P_{2,2}$
 - ▶ $R_{14}: \neg P_{1,3}$

Esempio di risoluzione

- ▶ Applichiamo l'eliminazione del bicondizionale a R_3 , seguita dal modus ponens con R_5
 - ▶ $R_{15}: P_{1,1} \vee P_{2,2} \vee P_{3,1}$
- ▶ Applichiamo la regola di risoluzione tra R_{15} e R_{13}

$$\frac{P_{1,1} \vee P_{2,2} \vee P_{3,1}, \quad \neg P_{2,2}}{P_{1,1} \vee P_{3,1}}$$

- ▶ $R_{16}: P_{1,1} \vee P_{3,1}$
- ▶ Applichiamo la regola di risoluzione tra R_{16} e R_1

$$\frac{P_{1,1} \vee P_{3,1}, \quad \neg P_{1,1}}{P_{3,1}}$$

- ▶ $R_{17}: P_{3,1}$

Dimostrazione per risoluzione

- ▶ Una dimostrazione di teoremi basato sulla risoluzione può, per ogni coppia di formule α e β della logica proposizionale, decidere se $\alpha \models \beta$
- ▶ Usa la forma normale congiuntiva (CNF - Conjunctive normal form)
 - ▶ Congiunzione di disgiunzioni di letterali (clausole)
- ▶ La regola di risoluzione è corretta
 - ▶ Si riescono a derivare soltanto sentenze implicate
- ▶ La risoluzione è completa nel senso che può essere sempre usata o per confermare o per rifiutare una sentenza (essa non può essere usata per enumerare le sentenze vere)

Forma Normale Congiuntiva

- ▶ La regola di risoluzione si applica soltanto a clausole (disgiunzione di letterali)
 - ▶ Come può essere considerata completa?
 - ▶ Si può utilizzare per l'intera logica proposizionale?

- ▶ Ogni formula della logica proposizionale è logicamente equivalente ad una congiunzione di clausole
 - ▶ Ogni formula in logica proposizionale può essere trasformata (convertita) in CNF

Un algoritmo di risoluzione

- ▶ L'algoritmo effettua una dimostrazione per assurdo, ovvero mostra che $KB \wedge \neg\alpha$ non è soddisfacibile
 - ▶ Converte $KB \wedge \neg\alpha$ in CNF
 - ▶ Si applica la regola di risoluzione alle clausole riguardanti
 - ▶ Ogni coppia che contiene letterali complementari è risolta per produrre una nuova clausola che viene aggiunta all'insieme (se non vi è già presente)
 - ▶ Il processo continua finché non si verifica una delle seguenti due possibilità:
 1. Non è possibile aggiungere alcuna clausola (α non è conseguenza logica di KB)
 2. La risoluzione applicata a due clausole dà come risultato la clausola **vuota** (KB consegue logicamente α), siamo arrivati ad una contraddizione

Un esempio di esecuzione

- ▶ Quando l'agente si trova in [1,1] non percepisce brezza
 - ▶ $R_2: B_{1,1} \Leftrightarrow (P_{1,2} \vee P_{2,1})$
 - ▶ $R_4: \neg B_{1,1}$
- ▶ Vogliamo dimostrare che $\alpha = \neg P_{2,1}$ è conseguenza di KB.
 - ▶ KB: $R_2 \wedge R_4$
 - ▶ Ricordiamo che R_2 può essere trasformata in CNF ottenendo $(\neg B_{1,1} \vee P_{1,2} \vee P_{2,1}) \wedge (\neg P_{1,2} \vee B_{1,1}) \wedge (\neg P_{2,1} \vee B_{1,1})$
 - ▶ $\alpha = \neg P_{2,1}$
 - ▶ Dobbiamo negare α e congiungerla a KB
- ▶ Clausole:
 - ▶ $(\neg B_{1,1} \vee P_{1,2} \vee P_{2,1}) \wedge (\neg P_{1,2} \vee B_{1,1}) \wedge (\neg P_{2,1} \vee B_{1,1}) \wedge \neg B_{1,1} \wedge P_{1,2}$

$\neg \alpha$



Un esempio di esecuzione

$(\neg B_{1,1} \vee P_{1,2} \vee P_{2,1})$

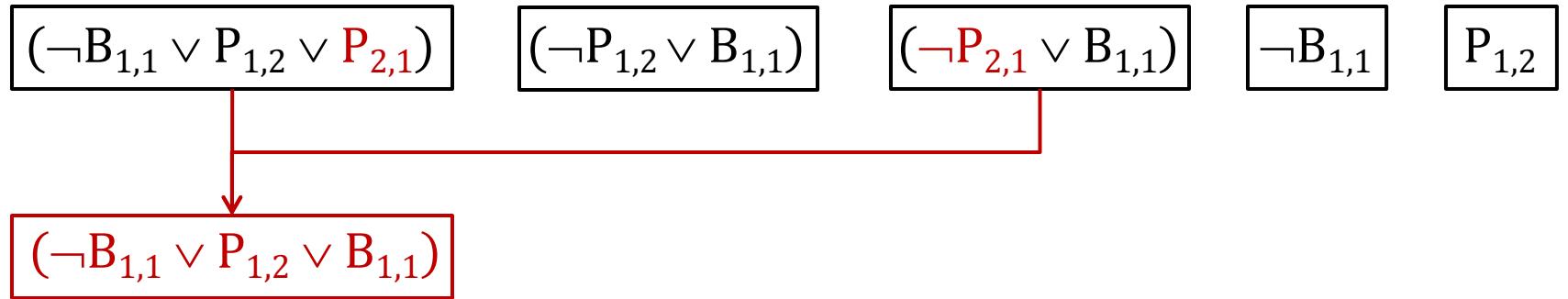
$(\neg P_{1,2} \vee B_{1,1})$

$(\neg P_{2,1} \vee B_{1,1})$

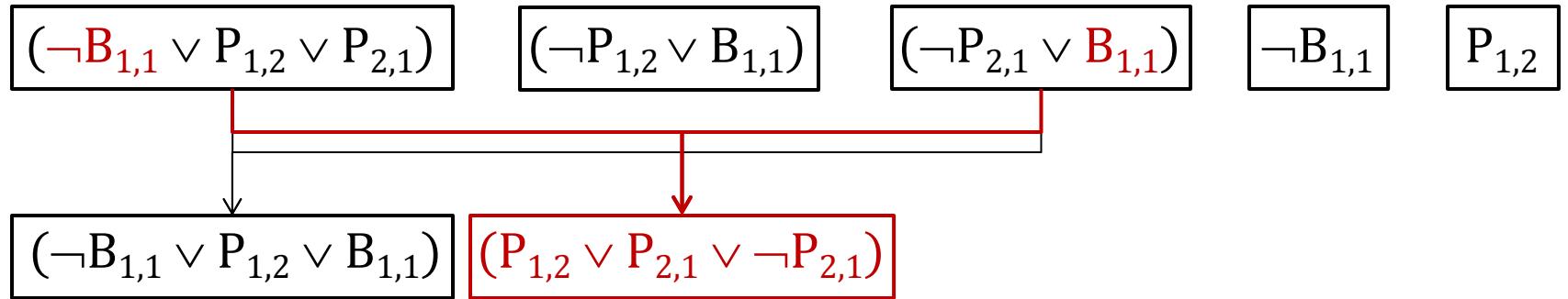
$\neg B_{1,1}$

$P_{1,2}$

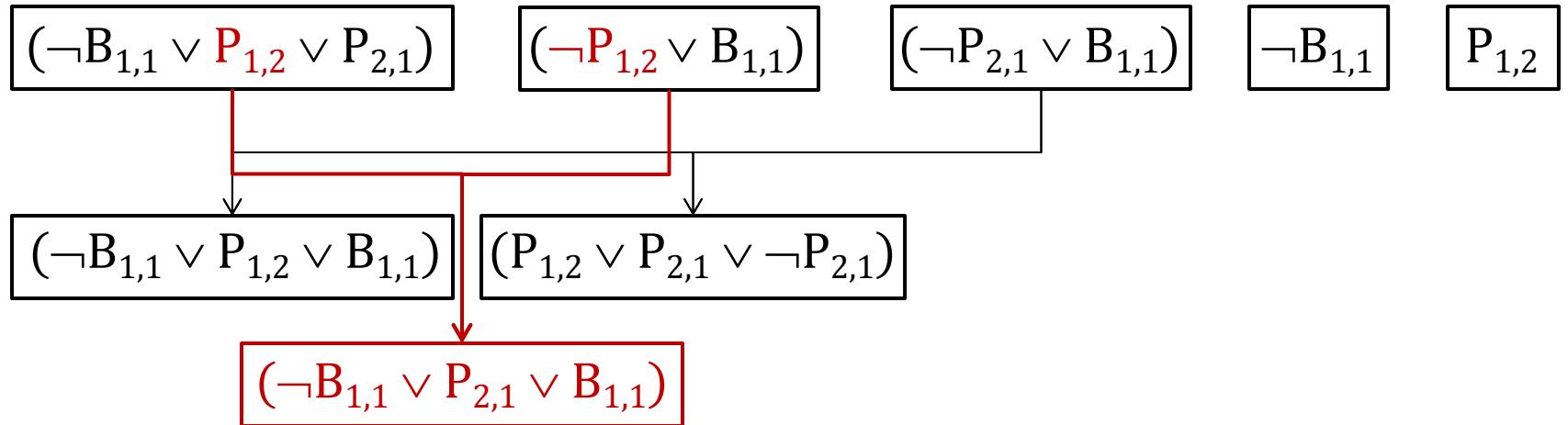
Un esempio di esecuzione



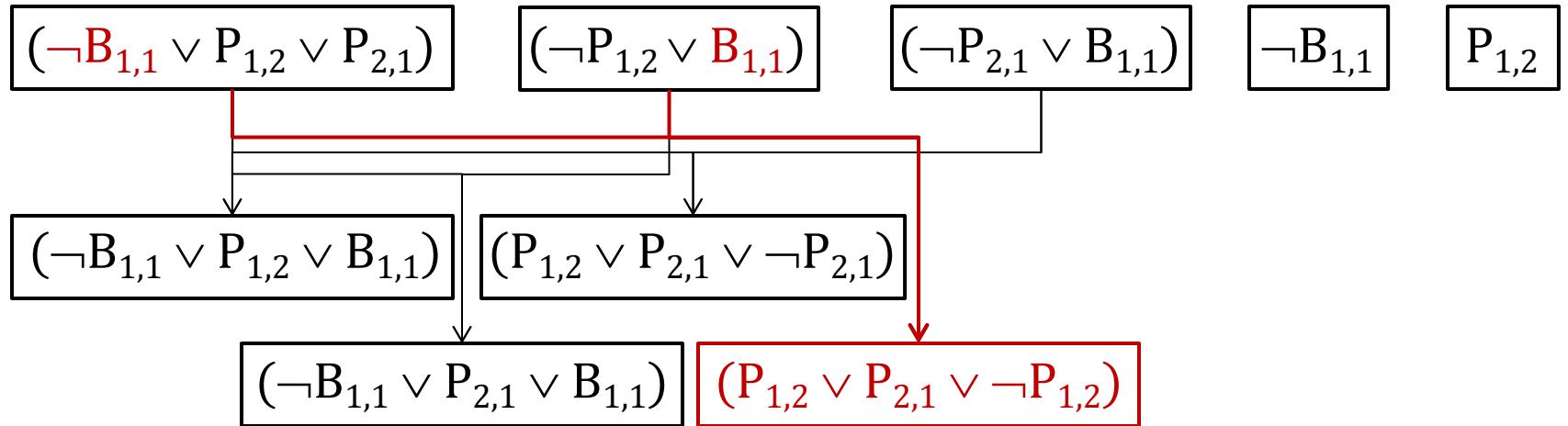
Un esempio di esecuzione



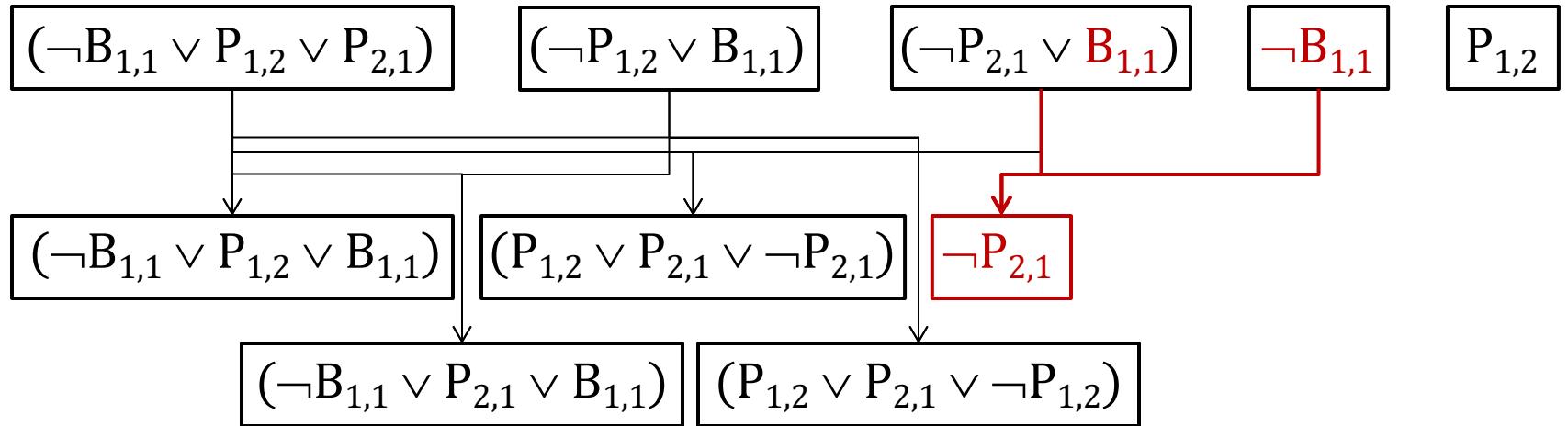
Un esempio di esecuzione



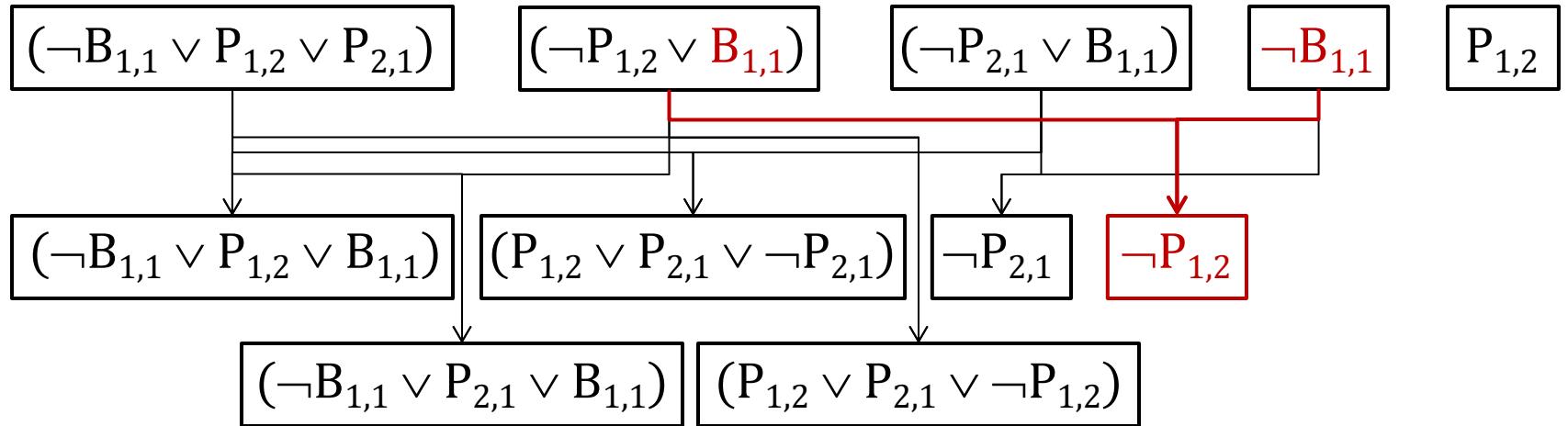
Un esempio di esecuzione



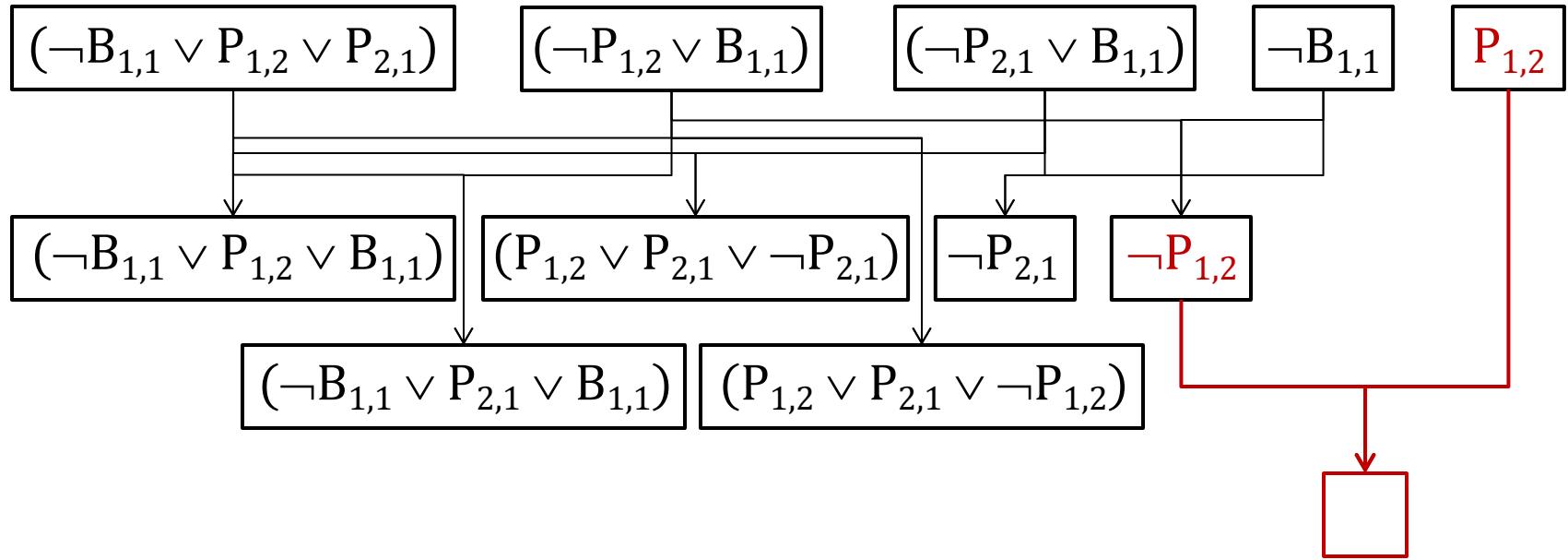
Un esempio di esecuzione



Un esempio di esecuzione



Un esempio di esecuzione



- ▶ È stato dimostrato l'assurdo, quindi si può dire che $\neg P_{1,2}$ è una conseguenza delle prime quattro clausole, ovvero KB.

Clausole speciali

- ▶ Alcune basi di conoscenza del mondo reale soddisfano certe restrizioni sulla forma delle formule che contengono (clausole)
 - ▶ **Clausola definita**
 - ▶ Una disgiunzione di letterali in cui *esattamente uno* è positivo
 - ▶ $(\neg L_{1,1} \vee \neg Brezza \vee B_{1,1})$ **SI** $(\neg B_{1,1} \vee P_{1,2} \vee P_{2,1})$ **NO**
 - ▶ **Clausola obiettivo**
 - ▶ Una disgiunzione di letterali in cui *nessuno* è positivo
 - ▶ $(\neg L_{1,1} \vee \neg Brezza \vee \neg B_{1,1})$ **SI**
 - ▶ **Clausola di Horn**
 - ▶ Una disgiunzione di letterali in cui *al massimo uno* è positivo
 - ▶ \Rightarrow una clausola di Horn è una clausola definita o una clausola obiettivo
 - ▶ Le clausole di Horn sono chiuse rispetto alla risoluzione
 - Se si risolvono due clausole di Horn, si ottiene una clausola di Horn

Clausole speciali

- ▶ Una grammatica per CNF, clausole di Horn e clausole definite

FormulaCNF → Clausola₁ ∧ ... ∧ Clausola_n

Clausola → Letterale₁ ∨ ... ∨ Letterale_m

Letterale → Simbolo | \neg Simbolo

Simbolo → P | Q | R | ...

FormaClausolaHorn → FormaClausolaDefinita | FormaClausolaObiettivo

FormaClausolaDefinita → (Simbolo₁ ∧ ... ∧ Simbolo_l) \Rightarrow Simbolo

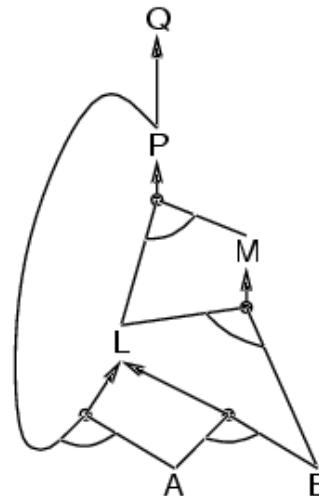
FormaClausolaObiettivo → (Simbolo₁ ∧ ... ∧ Simbolo_l) \Rightarrow \neg Simbolo

Clausole Definite/Horn: Proprietà

- ▶ Le basi di conoscenza contenenti soltanto clausole definite sono interessanti per tre ragioni
 1. Le clausole definite possono essere scritte come un'implicazione
 - ▶ $(\neg L_{1,1} \vee \neg \text{Brezza} \vee B_{1,1}) \rightarrow (L_{1,1} \wedge \text{Brezza}) \Rightarrow B_{1,1}$
 - ▶ Il singolo letterale $L_{1,1}$ viene chiamato **fatto**
 2. L'inferenza sulle clausole di Horn può essere svolta mediante gli algoritmi di **forward/backward chaining**
 - ▶ Questo tipo di inferenza è la base della programmazione logica
 3. Con le clausole di Horn è possibile determinare la conseguenza logica in tempo **lineare** rispetto alla dimensione della base di conoscenza

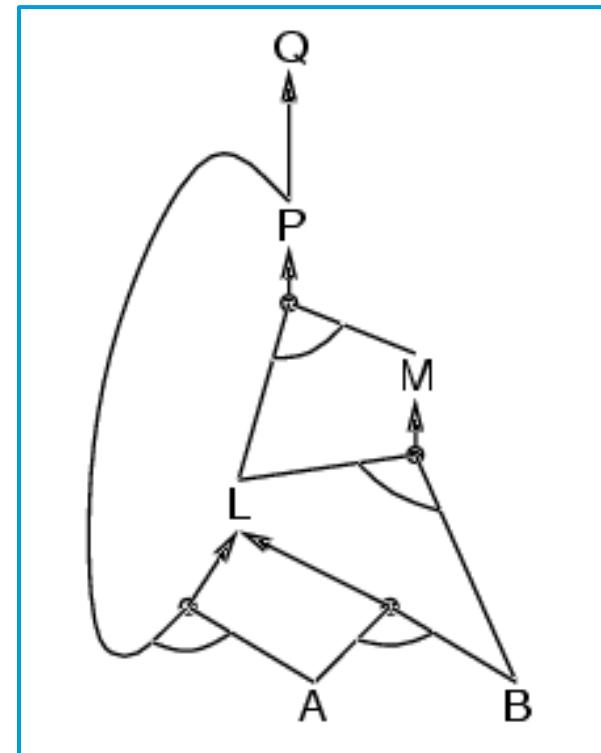
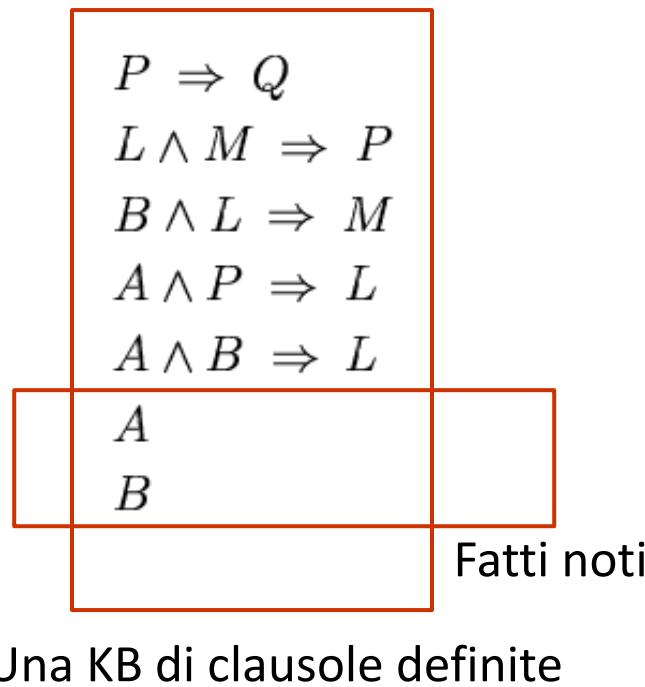
Forward Chaining

- ▶ L'algoritmo di concatenazione in avanti (forward chaining) determina se un singolo simbolo proposizionale q – la query – è conseguenza logica di una KB composta da clausole definite
 - ▶ IDEA: Per ogni clausola le cui premesse sono soddisfatte, aggiungi la sua conclusione all'insieme dei fatti noti, finché non viene aggiunta q o non è più possibile effettuare alcuna inferenza

$$\begin{aligned} P &\Rightarrow Q \\ L \wedge M &\Rightarrow P \\ B \wedge L &\Rightarrow M \\ A \wedge P &\Rightarrow L \\ A \wedge B &\Rightarrow L \\ A \\ B \end{aligned}$$


Un esempio di esecuzione

- ▶ Una semplice KB composta da clausole definite i cui fatti conosciuti sono A e B



Grafo AND-OR

Un esempio di esecuzione

- ▶ Una semplice KB composta da clausole definite i cui fatti conosciuti sono A e B

$$P \Rightarrow Q$$

$$\boxed{L \wedge M \Rightarrow P}$$

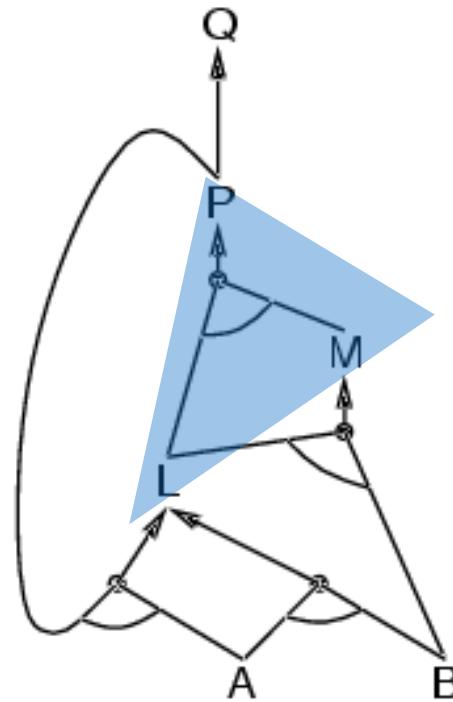
$$B \wedge L \Rightarrow M$$

$$A \wedge P \Rightarrow L$$

$$A \wedge B \Rightarrow L$$

$$A$$

$$B$$



Un esempio di esecuzione

- ▶ Una semplice KB composta da clausole definite i cui fatti conosciuti sono A e B

$$P \Rightarrow Q$$

$$L \wedge M \Rightarrow P$$

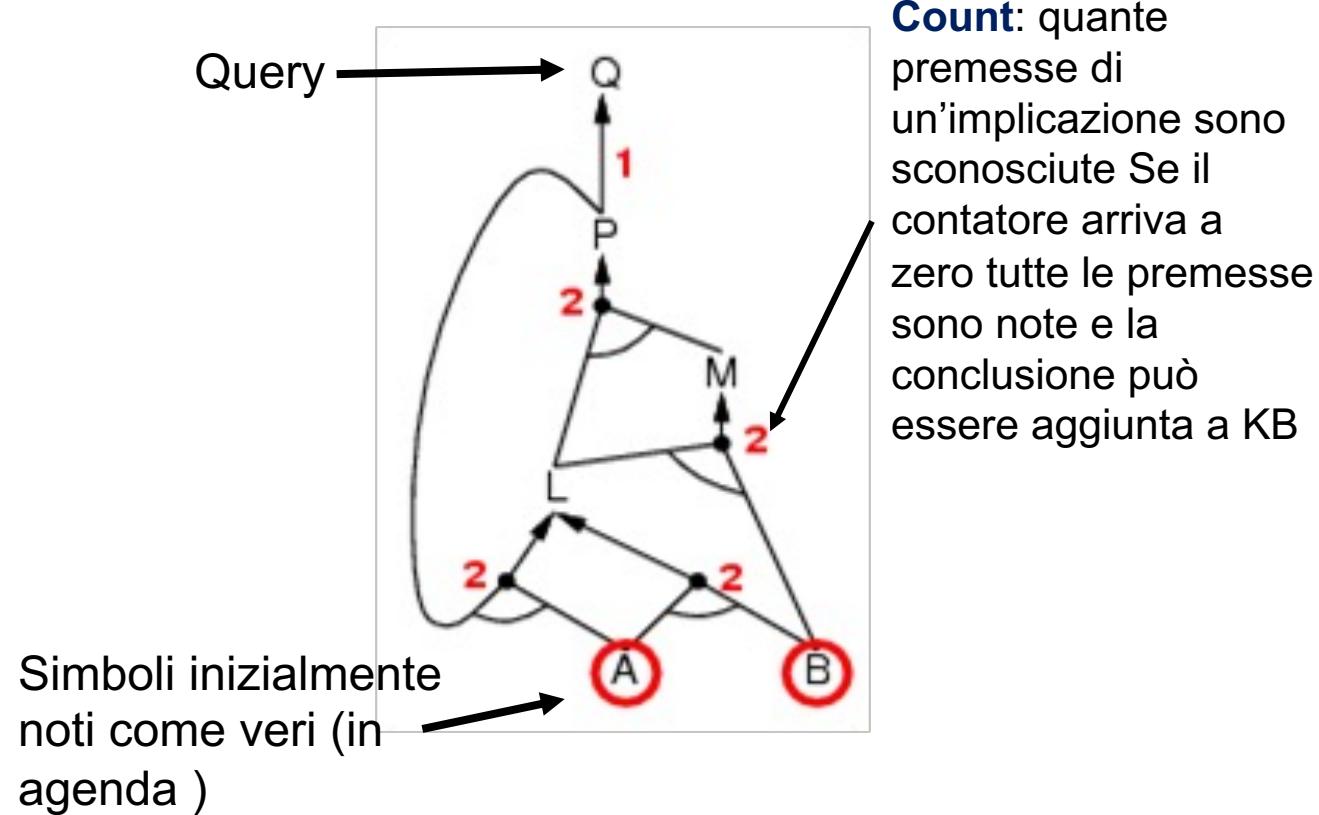
$$B \wedge L \Rightarrow M$$

$$A \wedge P \Rightarrow L$$

$$A \wedge B \Rightarrow L$$

A

B



Un esempio di esecuzione

- ▶ Una semplice KB composta da clausole definite i cui fatti conosciuti sono A e B

$$P \Rightarrow Q$$

$$L \wedge M \Rightarrow P$$

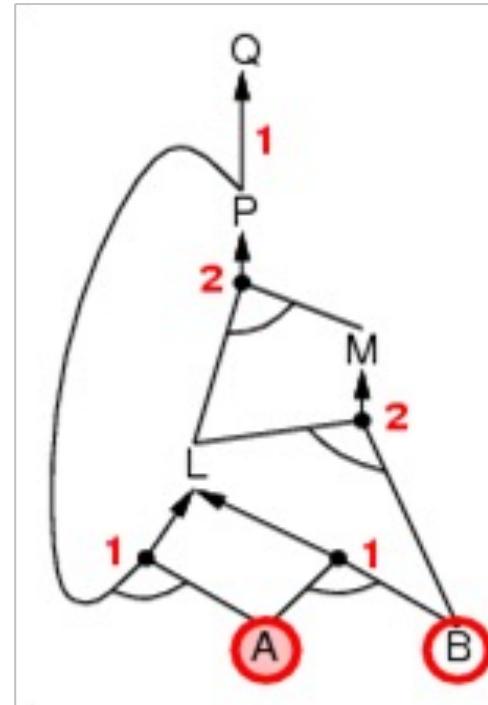
$$B \wedge L \Rightarrow M$$

$$A \wedge P \Rightarrow L$$

$$A \wedge B \Rightarrow L$$

$$A$$

$$B$$



Un esempio di esecuzione

- ▶ Una semplice KB composta da clausole definite i cui fatti conosciuti sono A e B

$$P \Rightarrow Q$$

$$L \wedge M \Rightarrow P$$

$$B \wedge L \Rightarrow M$$

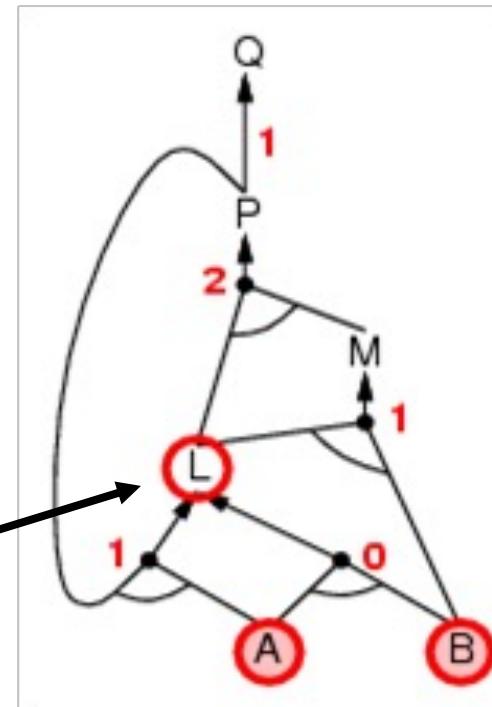
$$A \wedge P \Rightarrow L$$

$$A \wedge B \Rightarrow L$$

$$A$$

$$B$$

Simbolo
inferito (in
inferred)



Un esempio di esecuzione

- ▶ Una semplice KB composta da clausole definite i cui fatti conosciuti sono A e B

$$P \Rightarrow Q$$

$$L \wedge M \Rightarrow P$$

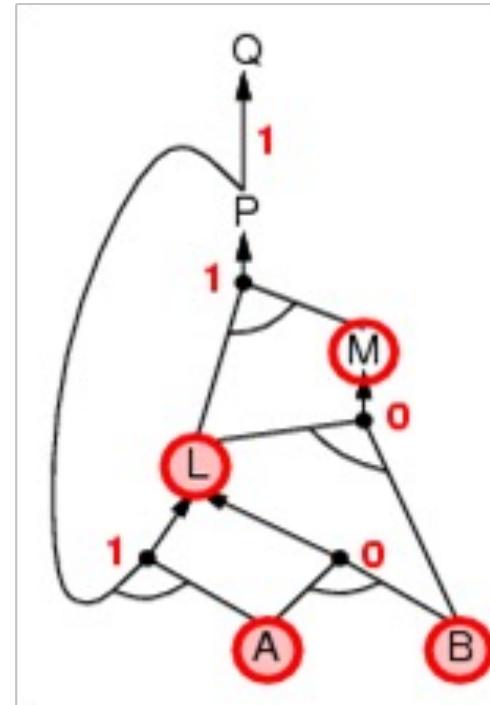
$$B \wedge L \Rightarrow M$$

$$A \wedge P \Rightarrow L$$

$$A \wedge B \Rightarrow L$$

$$A$$

$$B$$



Un esempio di esecuzione

- ▶ Una semplice KB composta da clausole definite i cui fatti conosciuti sono A e B

$$P \Rightarrow Q$$

$$L \wedge M \Rightarrow P$$

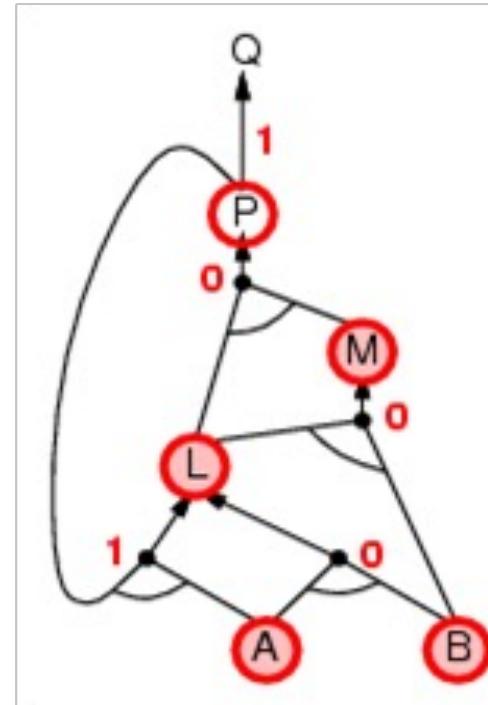
$$B \wedge L \Rightarrow M$$

$$A \wedge P \Rightarrow L$$

$$A \wedge B \Rightarrow L$$

$$A$$

$$B$$



Un esempio di esecuzione

- ▶ Una semplice KB composta da clausole definite i cui fatti conosciuti sono A e B

$$P \Rightarrow Q$$

$$L \wedge M \Rightarrow P$$

$$B \wedge L \Rightarrow M$$

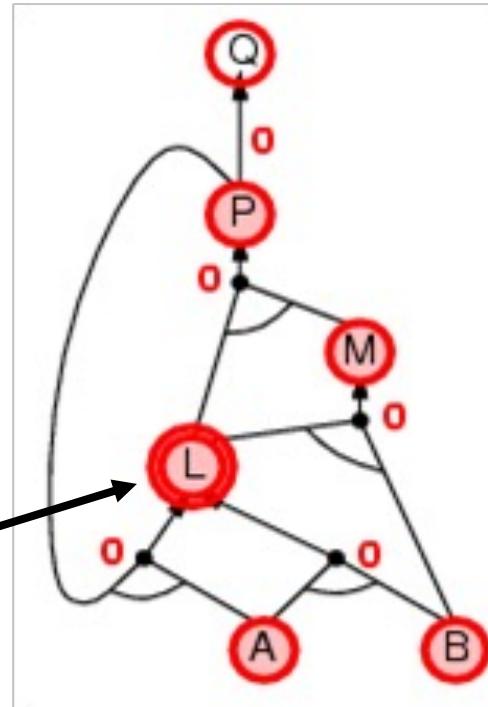
$$A \wedge P \Rightarrow L$$

$$A \wedge B \Rightarrow L$$

A

B

Simbolo già
processato. Non è
aggiunto a KB Evita
ridondanze e loop



Un esempio di esecuzione

- ▶ Una semplice KB composta da clausole definite i cui fatti conosciuti sono A e B

$$P \Rightarrow Q$$

$$L \wedge M \Rightarrow P$$

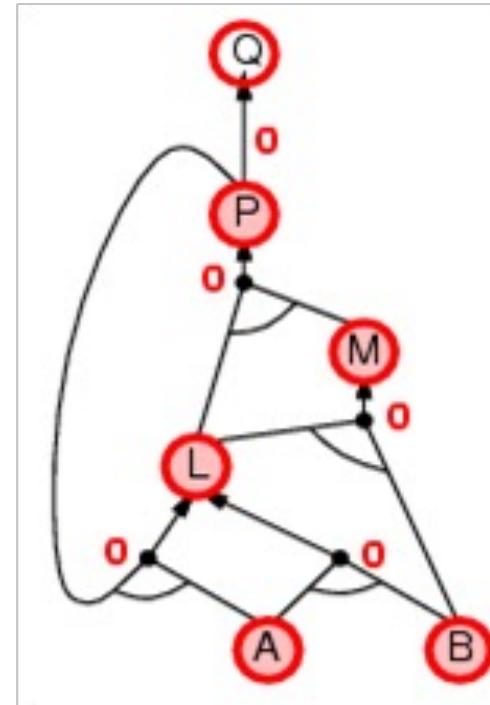
$$B \wedge L \Rightarrow M$$

$$A \wedge P \Rightarrow L$$

$$A \wedge B \Rightarrow L$$

$$A$$

$$B$$



Un esempio di esecuzione

- ▶ Una semplice KB composta da clausole definite i cui fatti conosciuti sono A e B

$$P \Rightarrow Q$$

$$L \wedge M \Rightarrow P$$

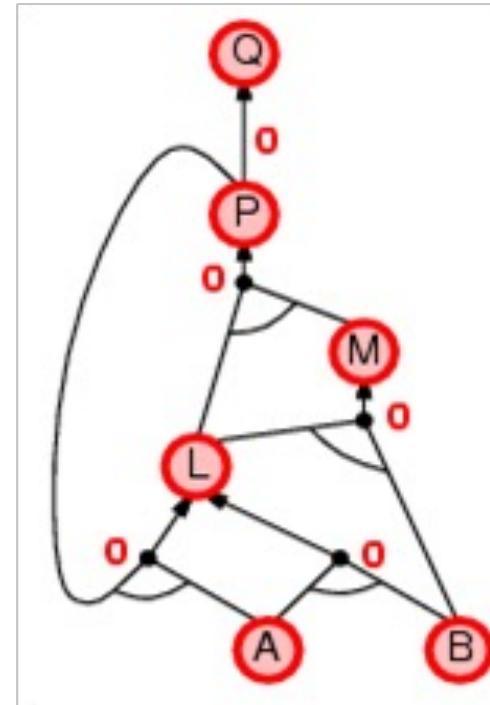
$$B \wedge L \Rightarrow M$$

$$A \wedge P \Rightarrow L$$

$$A \wedge B \Rightarrow L$$

$$A$$

$$B$$



Backward Chaining

- ▶ L'algoritmo di concatenazione all'indietro (backward chaining) determina se un singolo simbolo proposizionale q – la query – è conseguenza logica di una KB composta da clausole definite
 - ▶ IDEA: Se è già noto che q è vera non fa nulla, altrimenti verifica se tutte le premesse di qualche implicazione che ha q come conclusione sono vere attraverso backward chaining
- ▶ Evita i cicli:
 - ▶ verifica se un nuovo subgoal è stato già inserito nello stack degli obiettivi
- ▶ Evita la ripetizione del calcolo
 - ▶ verifica se un nuovo subgoal è stato già dimostrato vero, o è già fallito
- ▶ Anche in questo caso, un'implementazione efficiente verrà eseguita in tempo lineare

Un esempio di esecuzione

- ▶ Una semplice KB composta da clausole definite i cui fatti conosciuti sono A e B

$$P \Rightarrow Q$$

$$L \wedge M \Rightarrow P$$

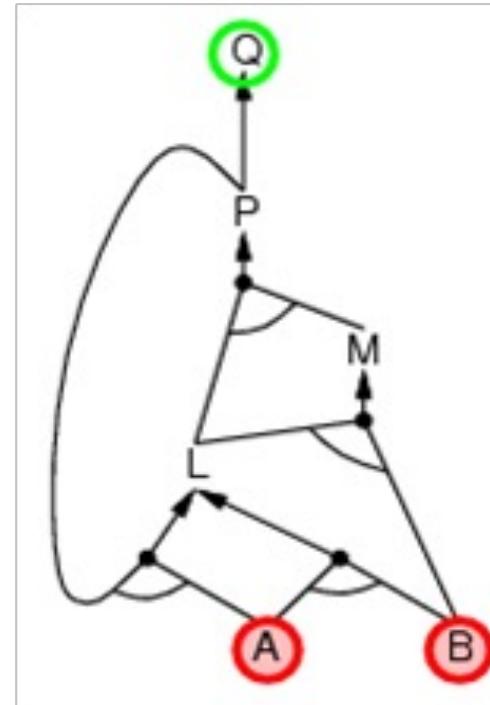
$$B \wedge L \Rightarrow M$$

$$A \wedge P \Rightarrow L$$

$$A \wedge B \Rightarrow L$$

$$A$$

$$B$$



Un esempio di esecuzione

- ▶ Una semplice KB composta da clausole definite i cui fatti conosciuti sono A e B

$$P \Rightarrow Q$$

$$L \wedge M \Rightarrow P$$

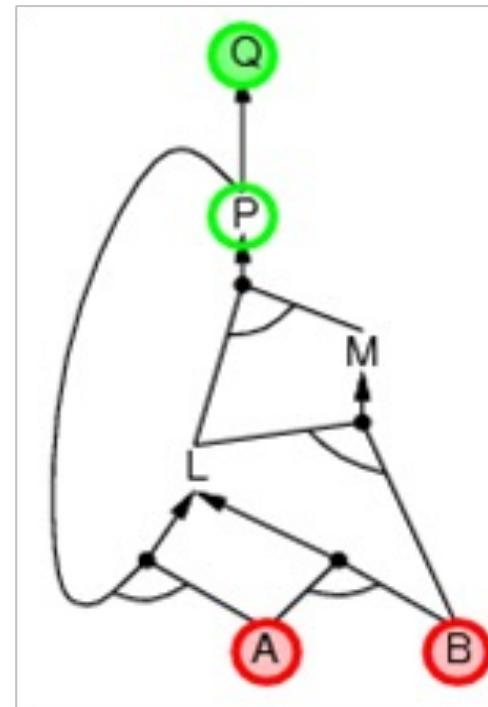
$$B \wedge L \Rightarrow M$$

$$A \wedge P \Rightarrow L$$

$$A \wedge B \Rightarrow L$$

$$A$$

$$B$$



Un esempio di esecuzione

- ▶ Una semplice KB composta da clausole definite i cui fatti conosciuti sono A e B

$$P \Rightarrow Q$$

$$L \wedge M \Rightarrow P$$

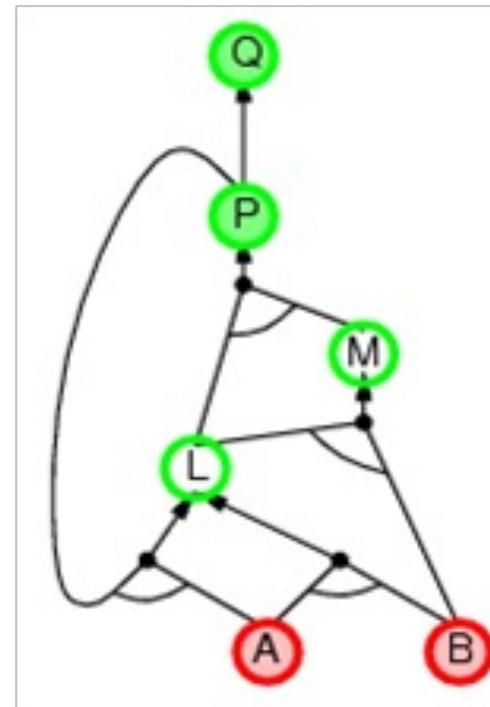
$$B \wedge L \Rightarrow M$$

$$A \wedge P \Rightarrow L$$

$$A \wedge B \Rightarrow L$$

$$A$$

$$B$$



Un esempio di esecuzione

- ▶ Una semplice KB composta da clausole definite i cui fatti conosciuti sono A e B

$$P \Rightarrow Q$$

$$L \wedge M \Rightarrow P$$

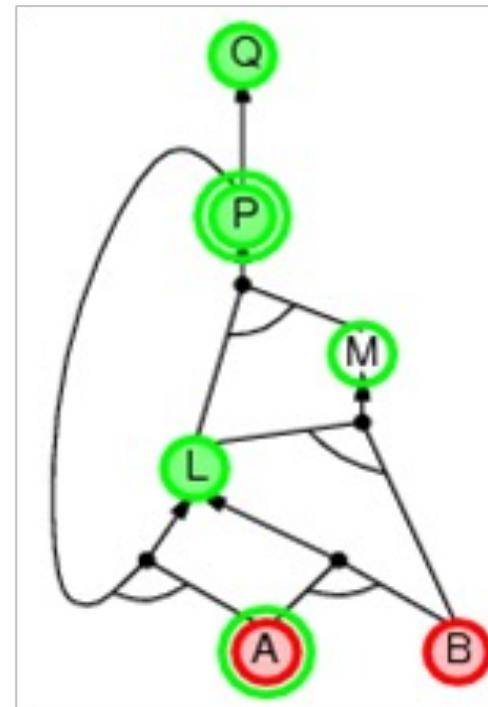
$$B \wedge L \Rightarrow M$$

$$A \wedge P \Rightarrow L$$

$$A \wedge B \Rightarrow L$$

$$A$$

$$B$$



Un esempio di esecuzione

- ▶ Una semplice KB composta da clausole definite i cui fatti conosciuti sono A e B

$$P \Rightarrow Q$$

$$L \wedge M \Rightarrow P$$

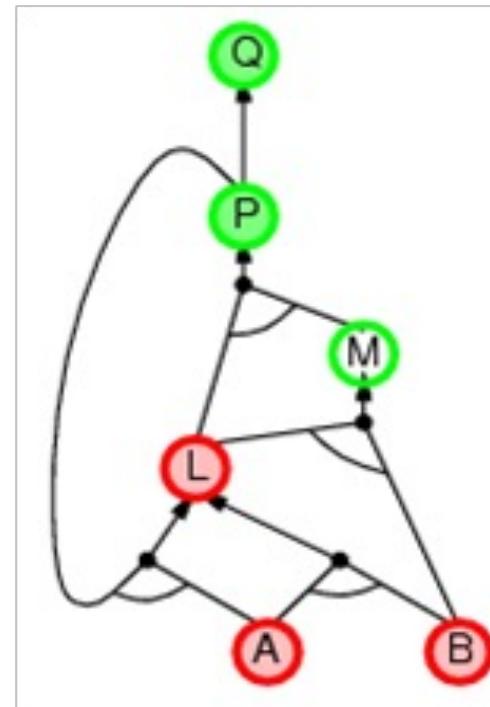
$$B \wedge L \Rightarrow M$$

$$A \wedge P \Rightarrow L$$

$$A \wedge B \Rightarrow L$$

$$A$$

$$B$$



Un esempio di esecuzione

- ▶ Una semplice KB composta da clausole definite i cui fatti conosciuti sono A e B

$$P \Rightarrow Q$$

$$L \wedge M \Rightarrow P$$

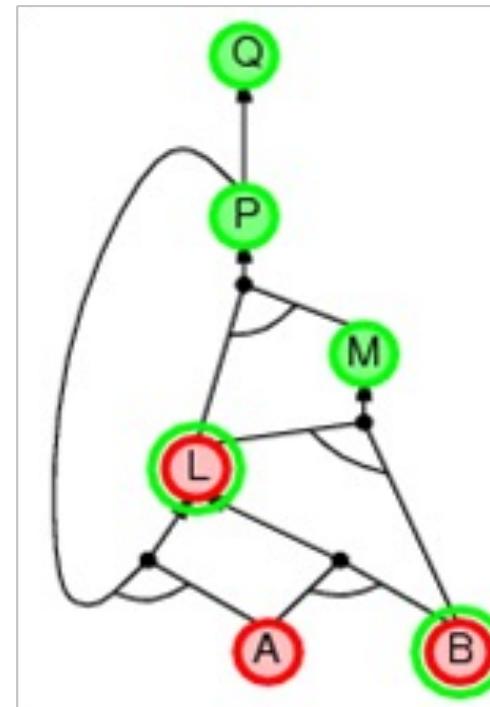
$$B \wedge L \Rightarrow M$$

$$A \wedge P \Rightarrow L$$

$$A \wedge B \Rightarrow L$$

$$A$$

$$B$$



Un esempio di esecuzione

- ▶ Una semplice KB composta da clausole definite i cui fatti conosciuti sono A e B

$$P \Rightarrow Q$$

$$L \wedge M \Rightarrow P$$

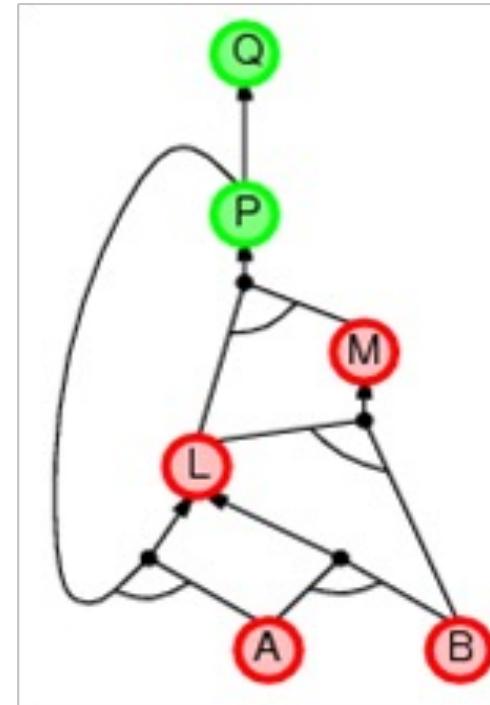
$$B \wedge L \Rightarrow M$$

$$A \wedge P \Rightarrow L$$

$$A \wedge B \Rightarrow L$$

$$A$$

$$B$$



Un esempio di esecuzione

- ▶ Una semplice KB composta da clausole definite i cui fatti conosciuti sono A e B

$$P \Rightarrow Q$$

$$L \wedge M \Rightarrow P$$

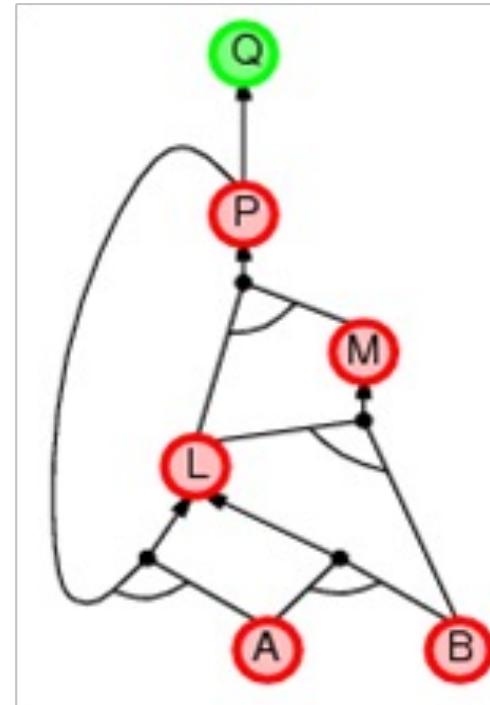
$$B \wedge L \Rightarrow M$$

$$A \wedge P \Rightarrow L$$

$$A \wedge B \Rightarrow L$$

$$A$$

$$B$$



Un esempio di esecuzione

- ▶ Una semplice KB composta da clausole definite i cui fatti conosciuti sono A e B

$$P \Rightarrow Q$$

$$L \wedge M \Rightarrow P$$

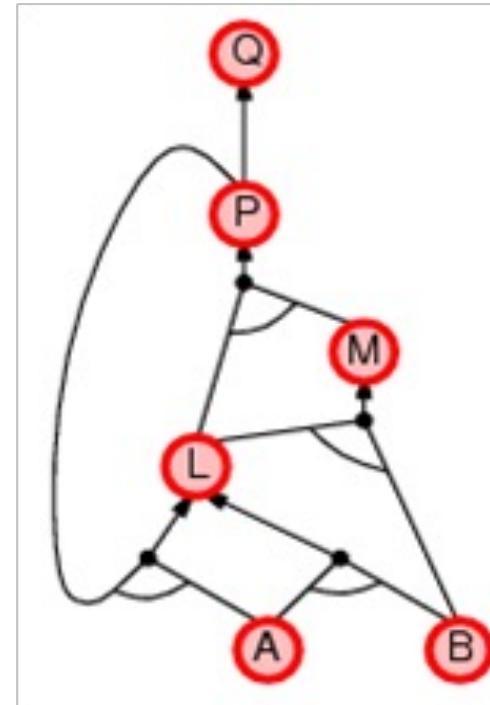
$$B \wedge L \Rightarrow M$$

$$A \wedge P \Rightarrow L$$

$$A \wedge B \Rightarrow L$$

$$A$$

$$B$$



Forward vs Backward Chaining

- ▶ Il forward chaining segue un **ragionamento guidato dai dati**
 - ▶ L'attenzione parte dai fatti conosciuti
 - ▶ Elaborazione automatica ed inconscia
 - ▶ Esempio di applicazione: object recognition
- ▶ Il backward chaining segue un **ragionamento basato sugli obiettivi**
 - ▶ Appropriato per il Problem-Solving
 - ▶ Esempio di applicazione: “Dove avrò lasciato le chiavi?”
 - ▶ Il costo è **molto meno** che lineare nelle dimensioni della KB

Agente basato sull'inferenza nel mondo Wumpus

- ▶ Un agente del mondo Wumpus che usa la logica proposizionale:
 - ▶ $\neg P_{1,1}$ (nessun pozzo in [1,1])
 - ▶ $\neg W_{1,1}$ (nessun Wumpus in [1,1])
 - ▶ $B_{x,y} \Leftrightarrow (P_{x,y+1} \vee P_{x,y-1} \vee P_{x+1,y} \vee P_{x-1,y})$ (Brezza vicino al pozzo)
 - ▶ $S_{x,y} \Leftrightarrow (W_{x,y+1} \vee W_{x,y-1} \vee W_{x+1,y} \vee W_{x-1,y})$ (Fetore vicino al Wumpus)
 - ▶ $W_{1,1} \vee W_{1,2} \vee \dots \vee W_{4,4}$ (almeno 1 Wumpus)
 - ▶ $\neg W_{1,1} \vee \neg W_{1,2}$ (al più 1 Wumpus)
 - ▶ $\neg W_{1,1} \vee \neg W_{8,9}$
 - ▶ ...
 - ▶ => 64 diversi simboli proposizionali, 155 sentenze

Limiti di espressività della Logica Proposizionale

- ▶ La logica proposizionale è un linguaggio semplice che consiste di simboli proposizionali e connettivi logici
 - ▶ Può gestire proposizioni che sono note come vere, note come false o indefinite
- ▶ Tuttavia, essa non è adatta ad ambienti di dimensione illimitata perché manca della potenza espressiva per affrontare in modo conciso tempo, spazio e schemi universali di relazioni tra oggetti
 - ▶ KB contiene sentenze fisiche per ogni cella
 - ▶ Proliferazione rapida di clausole



Intelligenza Artificiale

Logica del Primo Ordine



Outline

- ▶ Linguaggi di rappresentazione
- ▶ Cos'è la logica del primo ordine?
- ▶ Sintassi e Semantica
- ▶ Usare la logica del primo ordine
- ▶ Il mondo del Wumpus in logica del primo ordine
- ▶ Ingegneria della conoscenza

Vantaggi della logica proposizionale

- ▶ La logica proposizionale è un linguaggio **dichiarativo**
- ▶ Gestisce anche **informazioni parziali**
 - ▶ Grazie all'uso della disgiunzione e della negazione
- ▶ La logica proposizionale è **composizione**
 - ▶ Il significato di una formula è una funzione del significato delle sue parti
- ▶ La semantica della logica proposizionale è **indipendente dal contesto**

Svantaggi della logica proposizionale

- ▶ La logica proposizionale non ha la potenza espressiva per descrivere un ambiente con molti oggetti in modo conciso
- ▶ Ad esempio, non è possibile dire (formalizzare) in modo semplice l'espressione

*“Nelle stanze adiacenti a quelle che contengono
un pozzo si percepisce uno spostamento d'aria”*

- ▶ È necessario scrivere una sentenza per ogni cella
 - ▶ $B_{1,1} \Leftrightarrow (P_{1,2} \vee P_{2,1})$

Outline

- ▶ Linguaggi di rappresentazione
- ▶ Cos'è la logica del primo ordine?
- ▶ Sintassi e Semantica
- ▶ Usare la logica del primo ordine
- ▶ Il mondo del Wumpus in logica del primo ordine
- ▶ Ingegneria della conoscenza

Logica del primo ordine

- ▶ Partendo dai concetti presenti nella logica proposizionale
- ▶ Semantica dichiarativa e compositiva, indipendenza dal contesto e non ambiguità
- ▶ Costruire una **logica più espressiva**
- ▶ Prendendo in prestito metodi di rappresentazione del linguaggio naturale
- ▶ Evitando le sue limitazioni

Logica del primo ordine

- ▶ La logica proposizionale assume che il mondo contiene **fatti**
- ▶ La logica del primo ordine (First-Order Logic – FOL), come nel linguaggio naturale, assume che il mondo contiene
 - ▶ **Oggetti:** persone, case, numeri, colori, guerre, partite a baseball, ...
 - ▶ **Relazioni:** *unarie*, come rosso, tondo, falso, primo; o *n-arie* come fratello di, più grande di, all'interno di, parte di, avvenuto dopo, ...
 - ▶ **Funzioni:** padre di, migliore amico, uno più di, all'inizio di, ...

Outline

- ▶ Linguaggi di rappresentazione
- ▶ Cos'è la logica del primo ordine?
- ▶ Sintassi e Semantica
- ▶ Usare la logica del primo ordine
- ▶ Il mondo del Wumpus in logica del primo ordine
- ▶ Ingegneria della conoscenza

Modelli per la logica del primo ordine

- ▶ I modelli della logica proposizionale collegano simboli a valori di verità predefiniti
- ▶ I modelli della logica del primo ordine sono più interessanti
 - ▶ Contengono oggetti
- ▶ Il dominio di un modello è l'insieme degli oggetti che contiene
 - ▶ Gli elementi del dominio
 - ▶ Oggetti: persone, case, numeri, colori, guerre, partite a baseball, ...

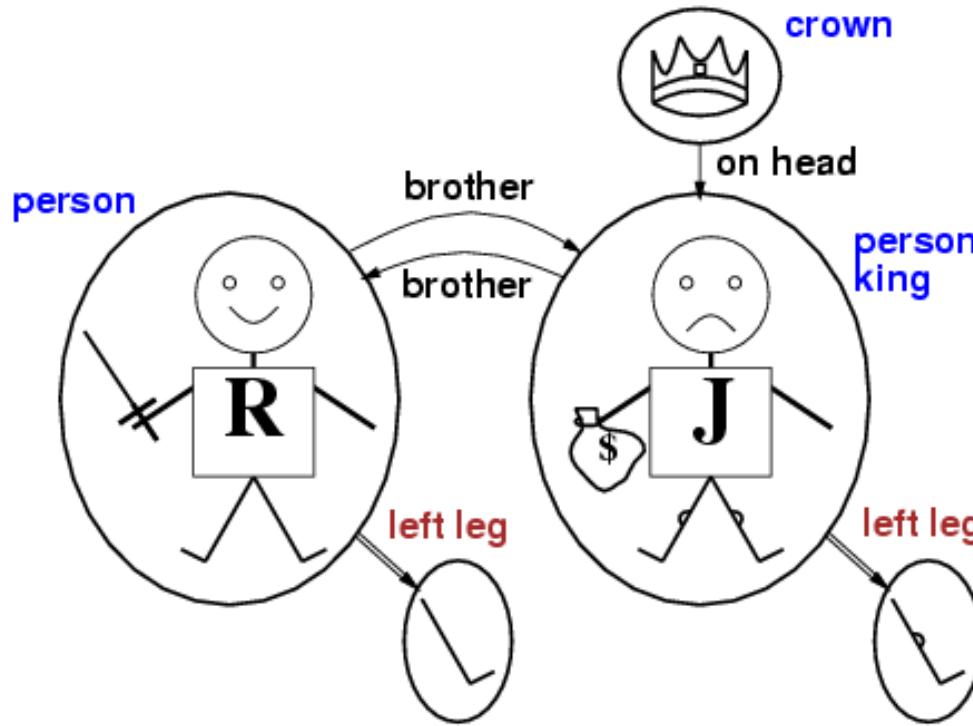
Modelli per la logica del primo ordine

- ▶ Gli oggetti del modello possono essere messi in relazione in molti modi
 - ▶ Relazioni: rosso, tondo, fratello di, all'interno di, parte di, ...
- ▶ Una relazione è un insieme di **tuple** di oggetti collegati
- ▶ Relazioni unarie (**proprietà**): si applicano ad un oggetto
 - ▶ Persona(Maria)
- ▶ Relazioni binarie: collegano coppie di oggetti
 - ▶ FratelloDi(Maria, Giuseppe)
- ▶ Relazioni n-arie: collegano n oggetti

Modelli per la logica del primo ordine

- ▶ Alcuni tipi di relazioni si possono considerare meglio come funzioni
 - ▶ Funzioni: padre di, migliore amico, uno più di, all'inizio di, ...
- ▶ Quando ogni dato oggetto può essere collegato attraverso di esse ad esattamente un altro oggetto
 - ▶ GambaSinistra(Maria)
 - ▶ GambaSinistra(Giuseppe)
- ▶ Nota: Non è necessario dare un nome alla gamba sinistra di Maria o di Giuseppe, dato che tutte le persone hanno una gamba sinistra

Modelli per la logica del primo ordine



- ▶ Oggetti: Riccardo, Giovanni, Corona
- ▶ Relazioni: FratelloDi(Riccardo,Giovanni), SullaTesta(Corona,Giovanni)
 - ▶ Proprietà (*Relazioni unarie*): Persona(Riccardo), Re(Giovanni),...
- ▶ Funzioni: GambaSinistra(Riccardo)

Elementi sintattici di base

► Linguaggio: il vocabolario

- ▶ Costanti Giovanni, Riccardo, A, ...
- ▶ Predicati Fratello, >, ...
- ▶ Funzioni RadiceQuadrata, GambaSinistra, ...
- ▶ Variabili x, y, a, b, ...
- ▶ Connettivi $\neg, \Rightarrow, \wedge, \vee, \Leftrightarrow$
- ▶ Uguaglianza =
- ▶ Quantificatori \forall, \exists

Termini

- ▶ Un **termine** è un'espressione logica che si riferisce ad un **oggetto**
- ▶ La sintassi dei termini ben formati:
 - ▶ Termine → Costante|Variabile|Funzione(Termine, ...)
(un numero di termini pari alla arità della funzione)
- ▶ Esempi di termini ben formati:
 - ▶ $x, A, B, 2$
 - ▶ $f(x, y)$
 - ▶ $+(2, 3)$
 - ▶ $\text{Padre-di(Giovanni)}$

Formula atomica

- ▶ Una **formula atomica** è composta da un simbolo di predicato seguito da una lista di termini tra parentesi
- ▶ La sintassi delle formule ben formate:
 - ▶ Formula-atomica → True | False | Termine = Termine |
Predicato(Termine, ...)
(un numero di termini pari alla arità del predicato)
- ▶ Esempi di formule atomiche ben formate:
 - ▶ Ama(Giorgio, Lucia)
 - ▶ +(2, 3) = 5
 - ▶ Amico(Padre-di(Giorgio), Padre-di(Elena))
- ▶ Una formula atomica è **vera** in un dato modello se la relazione a cui fa riferimento il simbolo predicato è verificata tra gli oggetti

Formula complessa

- ▶ Una **formula complessa** (o formula) è composta da formule atomiche che possono essere composte mediante i connettivi logici
- ▶ La sintassi delle formule ben formate:
 - ▶ $\text{Formula} \rightarrow \text{Formula-atomica} \mid \text{Formula Connuttivo}$
 $\text{Formula} \mid \text{Quantificatore Variabile Formula} \mid$
 $\neg \text{Formula} \mid (\text{Formula})$
- ▶ Esempi di formule ben formate:
 - ▶ $\text{Studia(Paolo)} \Rightarrow \text{Promosso(Paolo)}$
 - ▶ $\text{Fratello(Riccardo, Giovanni)} \wedge \text{Fratello(Giovanni, Riccardo)}$

Quantificatori

- ▶ Un **quantificatore** permette di esprimere proprietà di intere collezioni di oggetti invece di enumerare gli oggetti per nome
 - ▶ Universale: “Per ogni” \forall
 - ▶ Esistenziale: “Esiste” \exists

Quantificatore universale

- ▶ Formalmente il quantificatore universale ha la seguente forma
- ▶ $\forall <\text{Variabile}><\text{Formula}>$

Per ogni x, se x è un Re, allora x è una Persona

$$\forall x \text{ Re}(x) \Rightarrow \text{Persona}(x)$$

- ▶ Dal punto di vista della semantica
 - ▶ $\forall x A(x)$ è vera in un dato modello **m** se A è vera in **tutte le possibili interpretazioni estese** costruite a partire dall'interpretazione fornita in **m**

Quantificatore universale

- ▶ Dal punto di vista della semantica
 - ▶ Se il dominio è finito equivale a un grosso \wedge
 - ▶ $\forall x \text{ Mortale}(x)$
 - ▶ $\text{Mortale}(\text{Gino}) \wedge \text{Mortale}(\text{Pippo}) \wedge \dots$
 - ▶ Tipicamente, siccome difficilmente una proprietà è universale, \forall si usa insieme a \Rightarrow
 - ▶ $\forall x \text{ Persona}(x) \Rightarrow \text{Mortale}(x)$
- ▶ Un errore comune da evitare: usare \wedge come connettivo principale di \forall
 - ▶ $\forall x \text{ Persona}(x) \wedge \text{Mortale}(x)$
 - ▶ Significa ognuno è persona e ognuno è mortale

Quantificatore esistenziale

- ▶ Formalmente il quantificatore esistenziale ha la seguente forma
- ▶ $\exists <\text{Variabile}><\text{Sentenza}>$

Esiste una x corona tale che x è SullaTesta di Giovanni

$$\exists x \text{ Corona}(x) \wedge \text{SullaTesta}(x, \text{Giovanni})$$

- ▶ Dal punto di vista della semantica
 - ▶ $\exists x A(x)$ è vera in un dato modello m se A è vera in almeno una interpretazione estesa che assegna x ad un elemento del dominio

Quantificatore esistenziale

- ▶ Dal punto di vista della semantica
 - ▶ Se il dominio è finito equivale a un grosso √
 - ▶ $\exists x \text{ Persona}(x)$
 - ▶ $\text{Persona}(\text{Gino}) \vee \text{Persona}(\text{Pippo}) \vee \dots$
 - ▶ Tipicamente, \exists si usa con \wedge
 - ▶ $\exists x \text{ Persona}(x) \wedge \text{Speciale}(x)$
- ▶ Un errore comune da evitare: usare \Rightarrow come connettivo principale di \exists
 - ▶ $\exists x \text{ Persona}(x) \Rightarrow \text{Mortale}(x)$
 - ▶ Rappresenta una formula troppo debole!

Quantificatori annidati

- ▶ Quando i quantificatori sono dello stesso tipo si può scrivere il quantificatore una volta seguito da più variabili
 - ▶ $\forall x, y \text{ Fratello}(x, y) \Rightarrow \text{Consanguineo}(x, y)$
- ▶ Quando i quantificatori sono diversi **il loro ordine è importante:**
 - ▶ $\forall x (\exists y Ama(x, y))$ *Tutti amano qualcuno*
 - ▶ $\exists y (\forall x Ama(x, y))$ *Esiste qualcuno amato da tutti*
- ▶ Ambito dei quantificatori:

<i>ambito di y</i>	<i>ambito di x</i>
$\forall x (\exists y \text{ Ama}(x, y))$	$\forall x (\exists y \text{ Ama}(x, y))$

Outline

- ▶ Linguaggi di rappresentazione
- ▶ Cos'è la logica del primo ordine?
- ▶ Sintassi e Semantica
- ▶ Usare la logica del primo ordine
- ▶ Il mondo del Wumpus in logica del primo ordine
- ▶ Ingegneria della conoscenza

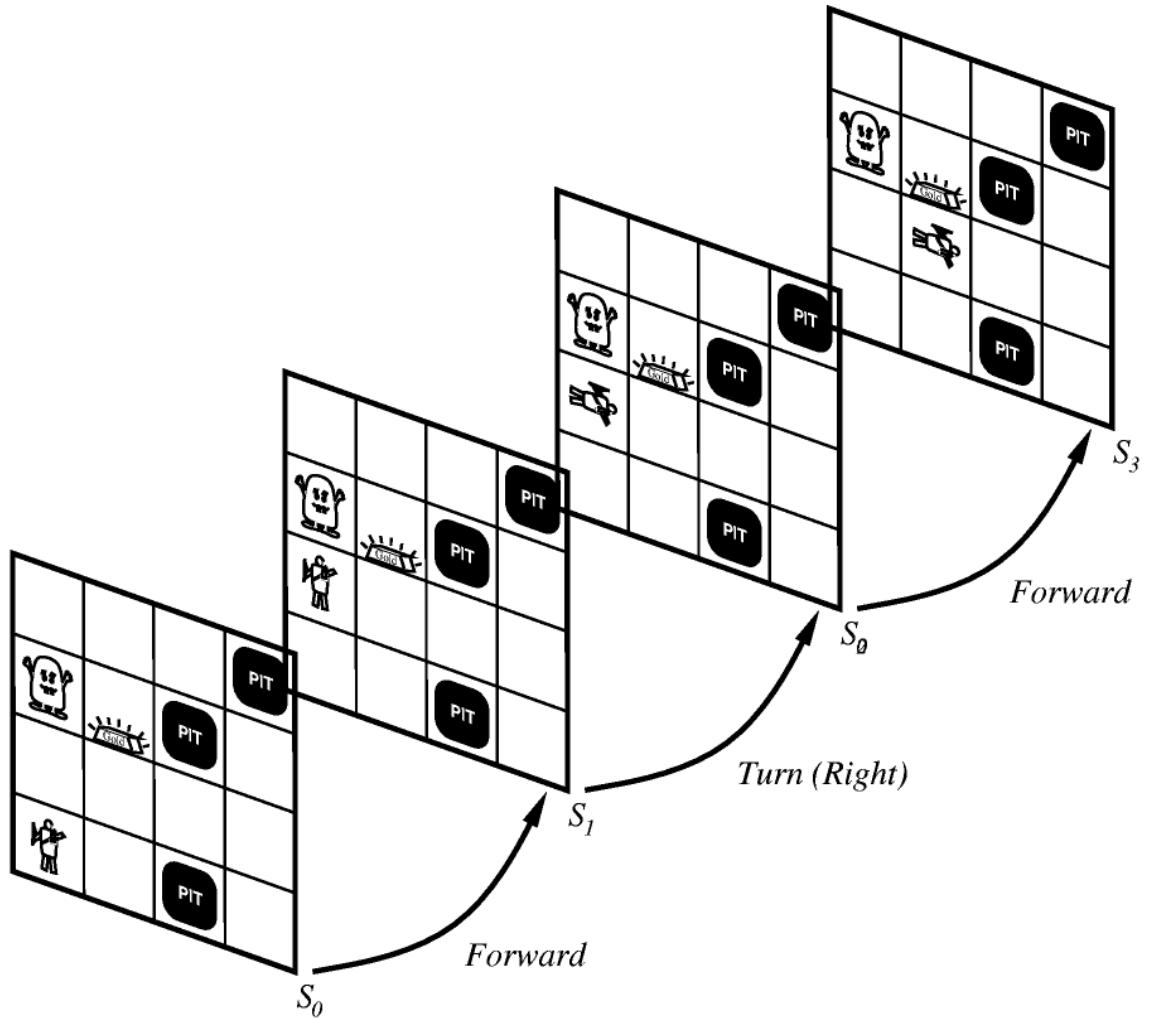
Il mondo del Wumpus

Termini Logici

- ▶ Gira(Destra)
- ▶ Gira(Sinistra)
- ▶ Avanti
- ▶ Afferra
- ▶ Scocca
- ▶ Esci

Costanti

- ▶ Fetore
- ▶ Brezza
- ▶ Scintillio
- ▶ Urto
- ▶ Urlo



Il mondo del Wumpus

- ▶ Le costanti sono raccolte in una lista
- ▶ Una tipica formula riguardante le percezioni sarà (predicato binario):
 - ▶ *Percezione([Fetore, Brezza, Scintillio, None, None], 5)*
- ▶ Per determinare l'azione migliore si esegue la query
 - ▶ AskVAR($\exists a \ BestAction(a, 5)$)
 - ▶ Restituisce una liste di legami come {*a/Afferra*}
- ▶ Le percezioni implicano alcuni fatti sullo stato corrente
 - ▶ $\forall t, s, g, m, c \ Percezione([s, Brezza, g, m, c], t) \Rightarrow Brezza(t)$
 - ▶ $\forall t, s, b, m, c \ Percezione([s, b, Scintillio, m, c], t) \Rightarrow Scintillio(t)$

Il mondo del Wumpus

- ▶ Semplici comportamenti “reattivi” possono essere implementati da formule di implicazione quantificata
 - ▶ $\forall t \text{ Scintillio}(t) \Rightarrow \text{BestAction}(\text{Afferra}, t)$
- ▶ Invece di usare nomi distinti per ogni locazione, usiamo un termine complesso in cui la riga e la colonna della locazione compaiono come indici interni (una lista). Le stanze adiacenti sono così definite
 - ▶ $\forall x,y,a,b \text{ Adiacente}([x,y], [a,b]) \Leftrightarrow ((x = a) \wedge (y = b-1 \vee y = b+1)) \vee ((y = b) \wedge (x = a-1 \vee x = a+1))$
- ▶ Usiamo un **predicato unario** per rappresentare il *Pozzo*
- ▶ Usiamo una **costante** per rappresentare il *Wumpus*

Il mondo del Wumpus

- ▶ Proprietà associate alle locazioni
- ▶ La posizione dell'agente cambia nel tempo
 - ▶ $Pos(\text{Agente}, s, t)$
 - ▶ L'agente si trova nella cella s all'istante t
- ▶ *Se l'agente in un determinato istante temporale sente fetore si trova in una cella puzzolente*
 - ▶ $\forall s, t \ Pos(\text{Agente}, s, t) \wedge \text{Fetore}(t) \Rightarrow \text{Puzzolente}(s)$
- ▶ *Se l'agente in un determinato istante temporale sente la brezza si trova in una cella ventosa*
 - ▶ $\forall s, t \ Pos(\text{Agente}, s, t) \wedge \text{Brezza}(t) \Rightarrow \text{Ventosa}(s)$

Il mondo del Wumpus

- ▶ Proprietà associate alle locazioni
- ▶ *Le locazioni ventose sono vicine ad un pozzo*
- ▶ **Regola diagnostica:** inferire la causa dall'effetto
 - ▶ $\forall s \text{ Ventosa}(s) \Rightarrow \exists r \text{ Adiacente}(r, s) \wedge \text{Pozzo}(r)$
- ▶ **Regola causale:** inferire l'effetto dalla causa (ragionamento basato sul modello)
 - ▶ $\forall r \text{ Pozzo}(r) \Rightarrow [\forall s \text{ Adiacente}(r, s) \Rightarrow \text{Ventosa}(s)]$
- ▶ Assioma per la freccia
 - ▶ $\forall t \text{ HaFreccia}(t+1) \Rightarrow (\text{HaFreccia}(t) \wedge \neg \text{Azione}(\text{Scocca}, t))$



Intelligenza Artificiale

**Inferenza nella
Logica del Primo Ordine**



Outline

- ▶ Inferenza del primo ordine
- ▶ Unificazione
- ▶ Modus Ponens generalizzato
- ▶ Forward/Backward chaining
- ▶ La programmazione logica
- ▶ Risoluzione

Inferenza del primo ordine

- ▶ Abbiamo già visto come sia possibile effettuare inferenze corrette e complete nella logica proposizionale
- ▶ Vogliamo estendere tali risultati ed ottenere **algoritmi corretti e completi** per KB in logica del primo ordine
 - ▶ Ottenendo una risposta (se questa esiste) a qualsiasi domanda espressa in logica del primo ordine
- ▶ Conversione in logica proposizionale
 - ▶ Sfruttando semplici regole per trasformare formule quantificate in formule prive di quantificatori
- ▶ Formulazione di metodi di inferenza in grado di manipolare direttamente formule del primo ordine

Regole di inferenza per \forall

- ▶ Istanziazione universale (\forall eliminazione)
- ▶ Sostituendo un termine ground (privo di variabili) alla variabile

$$\frac{\forall v \quad \alpha}{\text{SUBST}(\{v/g\}, \alpha)}$$



$\text{SUBST}(\theta, \alpha)$ applicazione della sostituzione θ alla formula α

per ogni variabile v e termine ground g

- ▶ Ad esempio vediamo come si possono ottenere nuove formule sostituendo: *tutti i re avidi sono malvagi*
 - ▶ $\forall x \text{Re}(x) \wedge \text{Avido}(x) \Rightarrow \text{Malvagio}(x)$
 - ▶ $\text{Re}(\text{Giovanni}) \wedge \text{Avido}(\text{Giovanni}) \Rightarrow \text{Malvagio}(\text{Giovanni})$
 $\text{Re}(\text{Riccardo}) \wedge \text{Avido}(\text{Riccardo}) \Rightarrow \text{Malvagio}(\text{Riccardo})$
 $\text{Re}(\text{Fratello}(\text{Giovanni})) \wedge \text{Avido}(\text{Fratello}(\text{Giovanni})) \Rightarrow \text{Malvagio}(\text{Fratello}(\text{Giovanni}))$
...

Regole di inferenza per \exists

- ▶ Istanziazione esistenziale (\exists eliminazione)
 - ▶ Sostituendo un nuovo simbolo costante (unico, che non appare in nessun'altra parte della KB) alla variabile

$$\frac{\exists v \quad \alpha}{\text{SUBST}(\{v/k\}, \alpha)}$$

per ogni sentenza α , variabile v e costante k

- ▶ Ad esempio vediamo come si può ottenere una nuova formula sostituendo: *Giovanni ha una corona sulla testa*
 - ▶ $\exists x \text{ Corona}(x) \wedge \text{SullaTesta}(x, \text{Giovanni})$
 - ▶ $\text{Corona}(C_1) \wedge \text{SullaTesta}(C_1, \text{Giovanni})$
- ▶ Il nuovo simbolo costante introdotto nella sostituzione è detto costante di Skolem

Equivalenza

- ▶ L'istanziazione universale può essere applicata più volte per aggiungere nuove sentenze
 - ▶ La nuova KB è **logicamente equivalente** a quella originale
- ▶ L'istanziazione esistenziale può essere applicata una sola volta per sostituire la sentenza esistenziale
 - ▶ La nuova KB non è logicamente equivalente alla vecchia, ma risulta essere soddisfacibile (vera in almeno un modello) se e soltanto se la KB originale lo era (**inferenzialmente equivalente**)
 - ▶ L'algoritmo di Skolem non mantiene l'equivalenza semantica.

Riduzione all'inferenza proposizionale

- ▶ Utilizzando le regole viste prima, diventa possibile ridurre l'inferenza del primo ordine a quella proposizionale
- ▶ Supponiamo che la KB contenga
 - ▶ $\forall x \text{Re}(x) \wedge \text{Avido}(x) \Rightarrow \text{Malvagio}(x)$
 - $\text{Re}(\text{Giovanni})$
 - $\text{Avido}(\text{Giovanni})$
 - $\text{Fratello}(\text{Riccardo}, \text{Giovanni})$
- ▶ Istanziando la formula universale **in tutti i modi possibili** otteniamo
 - ▶ $\text{Re}(\text{Giovanni}) \wedge \text{Avido}(\text{Giovanni}) \Rightarrow \text{Malvagio}(\text{Giovanni})$
 - $\text{Re}(\text{Riccardo}) \wedge \text{Avido}(\text{Riccardo}) \Rightarrow \text{Malvagio}(\text{Riccardo})$
 - $\text{Re}(\text{Giovanni})$
 - $\text{Avido}(\text{Giovanni})$
 - $\text{Fratello}(\text{Riccardo}, \text{Giovanni})$

Proposizionalizzazione

- ▶ La tecnica appena applicata è detta **proposizionalizzazione**
 - ▶ Può essere applicata in modo assolutamente generale
- ▶ Ad esempio i simboli proposizionali ottenuti nell'esempio precedente sono
 - ▶ *Re(Giovanni), Avido(Giovanni), Malvagio(Giovanni), Re(Riccardo), ...*
- ▶ A questo punto possiamo trattare la KB come proposizionale e applicare gli algoritmi visti
- ▶ Problema: le costanti sono in numero finito, ma se ci sono **funzioni?**
- ▶ Se la KB contiene il simbolo funzione *Padre*
 - ▶ Il numero di istanze da creare è infinito!
 - ▶ *Giovanni, Padre(Giovanni), Padre(Padre(Giovanni)) ...*

Teorema di Herbrand (1930)

- ▶ Se una formula α segue logicamente dalla KB originale (quella in logica del primo ordine), allora ci sarà una dimostrazione che coinvolge solo un sottoinsieme **finito** della KB proposizionalizzata
- ▶ Si può procedere incrementalmente ...
 1. Creare le istanze con le costanti
 2. Creare poi tutti i termini di profondità 1
 - ▶ Padre(*Giovanni*), Madre(*Giovanni*)
 3. Poi quelle con profondità 2
 - ▶ Padre(Padre(*Giovanni*)), Padre(Madre(*Giovanni*)) ...
 4. Finché la formula α non è conseguenza logica della KB costruita fino a quel momento
- ▶ L'algoritmo è **completo**
 - ▶ ogni formula che segue logicamente può essere dimostrata

Semidecidibilità in FOL

- ▶ Cosa succede quando una formula non è conseguenza logica della KB?
 - ▶ se $\text{KB} \not\models \alpha$ il processo non termina!
- ▶ Teorema: Turing (1936) e Church (1936)

*Il problema della conseguenza logica, per la logica del primo ordine è **semidecidibile**: questo significa che esistono algoritmi che rispondono affermativamente per ogni formula che è conseguenza logica, ma nessun algoritmo potrà rispondere negativamente per ogni formula che non è conseguenza logica*

Problemi con la proposizionalizzazione

- ▶ Sembra che la proposizionalizzazione generi una grossa quantità di sentenze irrilevanti
- ▶ Ad esempio considerando la seguente KB
 - ▶ $\forall x \text{Re}(x) \wedge \text{Avido}(x) \Rightarrow \text{Malvagio}(x)$
 - $\text{Re}(\text{Giovanni})$
 - $\forall y \text{Avido}(y)$
 - $\text{Fratello}(\text{Riccardo}, \text{Giovanni})$
- ▶ Sembra ovvio che si riesca ad ottenere **Malvagio(Giovanni)**, tuttavia vengono prodotti una serie di fatti irrilevanti, come **Avido(Riccardo)**
 - ▶ Con p predicati di arità k ed n costanti, ci saranno $p \cdot n^k$ istanziazioni!

Outline

- ▶ Inferenza del primo ordine
- ▶ Unificazione
- ▶ Modus Ponens generalizzato
- ▶ Forward/Backward chaining
- ▶ La programmazione logica
- ▶ Risoluzione

Unificazione

- ▶ Il processo di **unificazione** determina se due espressioni possono essere rese identiche mediante una sostituzione di termini alle variabili
- ▶ È un componente chiave di tutti gli algoritmi di inferenza del primo ordine
- ▶ L'algoritmo UNIFY prende due formule e, se esiste, restituisce un loro **unificatore**
 - ▶ $\text{UNIFY}(p, q) = \theta$ se $\text{SUBST}(\theta, p) = \text{SUBST}(\theta, q)$
 - ▶ Possiamo inferire che esiste una sostituzione θ per la quale $\text{Re}(x)$ e $\text{Avido}(x)$ corrispondano a $\text{Re}(\text{Giovanni})$ e $\text{Avido}(y)$
 - ▶ $\theta = \{x/\text{Giovanni}, y/\text{Giovanni}\}$ funziona!

Unificazione

- ▶ Vediamo qualche esempio di come si dovrebbe comportare UNIFY
- ▶ Supponiamo di avere una query: *Chi conosce Giovanni?*

Conosce(Giovanni, x)

p	q	θ
<i>Conosce(Giovanni, x)</i>	<i>Conosce(Giovanni, Giacomina)</i>	
<i>Conosce(Giovanni, x)</i>	<i>Conosce(y, Guglielmo)</i>	
<i>Conosce(Giovanni, x)</i>	<i>Conosce(y, Madre(y))</i>	
<i>Conosce(Giovanni, x)</i>	<i>Conosce(x, Elisabetta)</i>	

Unificazione

- ▶ Vediamo qualche esempio di come si dovrebbe comportare UNIFY
- ▶ Supponiamo di avere una query: *Chi conosce Giovanni?*

Conosce(Giovanni, x)

p	q	θ
<i>Conosce(Giovanni, x)</i>	<i>Conosce(Giovanni, Giacomina)</i>	$\{x/Giacomina\}$
<i>Conosce(Giovanni, x)</i>	<i>Conosce(y, Guglielmo)</i>	
<i>Conosce(Giovanni, x)</i>	<i>Conosce(y, Madre(y))</i>	
<i>Conosce(Giovanni, x)</i>	<i>Conosce(x, Elisabetta)</i>	

Unificazione

- ▶ Vediamo qualche esempio di come si dovrebbe comportare UNIFY
- ▶ Supponiamo di avere una query: *Chi conosce Giovanni?*

Conosce(Giovanni, x)

p	q	θ
<i>Conosce(Giovanni, x)</i>	<i>Conosce(Giovanni, Giacomina)</i>	{x/Giacomina}
<i>Conosce(Giovanni, x)</i>	<i>Conosce(y, Guglielmo)</i>	{x/Guglielmo, y/Giovanni}
<i>Conosce(Giovanni, x)</i>	<i>Conosce(y, Madre(y))</i>	
<i>Conosce(Giovanni, x)</i>	<i>Conosce(x, Elisabetta)</i>	

Unificazione

- ▶ Vediamo qualche esempio di come si dovrebbe comportare UNIFY
- ▶ Supponiamo di avere una query: *Chi conosce Giovanni?*

Conosce(Giovanni, x)

p	q	θ
<i>Conosce(Giovanni, x)</i>	<i>Conosce(Giovanni, Giacomina)</i>	{x/Giacomina}
<i>Conosce(Giovanni, x)</i>	<i>Conosce(y, Guglielmo)</i>	{x/Guglielmo, y/Giovanni}
<i>Conosce(Giovanni, x)</i>	<i>Conosce(y, Madre(y))</i>	{y/Giovanni, x/Madre(Giovanni)}
<i>Conosce(Giovanni, x)</i>	<i>Conosce(x, Elisabetta)</i>	

Unificazione

- ▶ Vediamo qualche esempio di come si dovrebbe comportare UNIFY
- ▶ Supponiamo di avere una query: *Chi conosce Giovanni?*

Conosce(Giovanni, x)

p	q	θ
<i>Conosce(Giovanni, x)</i>	<i>Conosce(Giovanni, Giacomina)</i>	{x/Giacomina}
<i>Conosce(Giovanni, x)</i>	<i>Conosce(y, Guglielmo)</i>	{x/Guglielmo, y/Giovanni}
<i>Conosce(Giovanni, x)</i>	<i>Conosce(y, Madre(y))</i>	{y/Giovanni, y/Madre(Giovanni)}
<i>Conosce(Giovanni, x)</i>	<i>Conosce(x, Elisabetta)</i>	<i>fallimento</i>

- ▶ L'ultima unificazione fallisce perché x non può assumere contemporaneamente i valori Giovanni ed Elisabetta

Standardizzazione separata

- ▶ Tutto ciò può essere evitato rinominando le variabili di una delle formule per evitare collisioni
 - ▶ Questa operazione prende il nome di **Standardizzazione Separata**
- ▶ Nell'esempio si può rinominare la x di $\text{Conosce}(x, \text{Elisabella})$ in z_{17} senza modificare il suo significato
$$\text{UNIFY}(\text{Conosce}(\text{Giovanni}, x), \text{Conosce}(z_{17}, \text{Elisabella})) = \{x/\text{Elisabella}, z_{17}/\text{Giovanni}\}$$
- ▶ In questo modo l'unificazione funziona!

Outline

- ▶ Inferenza del primo ordine
- ▶ Unificazione
- ▶ Modus Ponens generalizzato
- ▶ Forward/Backward chaining
- ▶ La programmazione logica
- ▶ Risoluzione

Modus Ponens Generalizzato (GMP)

- ▶ Per tutte le formule atomiche p_i , p'_i e q , quando esiste una sostituzione θ tale che $\text{SUBST}(\theta, p'_i) = \text{SUBST}(\theta, p_i)$ per tutti gli i , si può applicare la regola di inferenza chiamata **modus ponens generalizzato (GMP)**

$$\frac{p'_1, p'_2, \dots, p'_n, (p_1 \wedge p_2 \wedge \dots \wedge p_n \Rightarrow q)}{\text{SUBST}(\theta, q)}$$

- ▶ Applicando tale regola al nostro esempio

p'_1 è $\text{Re}(Giovanni)$

p_1 è $\text{Re}(x)$

p'_2 è $\text{Avido}(y)$

p_2 è $\text{Avido}(x)$

θ è $\{x/Giovanni, y/Giovanni\}$

q è $\text{Malvagio}(x)$

$\text{SUBST}(\theta, q)$ è $\text{Malvagio}(Giovanni)$

Outline

- ▶ Inferenza del primo ordine
- ▶ Unificazione
- ▶ Modus Ponens generalizzato
- ▶ Forward/Backward chaining
- ▶ La programmazione logica
- ▶ Risoluzione

Forward Chaining

- ▶ Abbiamo già visto un algoritmo di **forward chaining** (**concatenazione in avanti**) per clausole definite proposizionali
 - ▶ Si parte con le formule atomiche nella KB
 - ▶ Si applica Modus Ponens in avanti
 - ▶ Si aggiungono nuove formule atomiche, finché non è più possibile compiere altre inferenze
- ▶ Vedremo come si può applicare l'algoritmo a clausole definite del primo ordine
- ▶ Clausole definite come *Situazione* \Rightarrow *Risposta* sono particolarmente utili per sistemi che effettuano inferenze non appena ricevono nuove informazioni

Clausole definite del primo ordine

- ▶ Le **clausole definite** del primo ordine sono molto simili a quelle proposizionali
 - ▶ Disgiunzioni di letterali dove esattamente uno dei quali è positivo
- ▶ Una clausola definita può essere
 - ▶ Una formula atomica
 - ▶ $\text{Re}(\text{Giovanni})$
 - ▶ Un'implicazione in cui l'antecedente è una congiunzione di letterali positivi e la conseguenza è un singolo letterale positivo
 - ▶ $\text{Re}(x) \wedge \text{Avido}(x) \Rightarrow \text{Malvagio}(x)$

Clausole definite del primo ordine

- ▶ A differenza delle clausole definite proposizionali, quelle del primo ordine possono includere variabili
 - ▶ Le variabili si considerano sempre quantificate universalmente
- ▶ Non tutte le KB possono essere convertite in un insieme di clausole definite
 - ▶ A causa della restrizione sulla presenza di un solo letterale positivo
- ▶ Tuttavia questo è possibile in molti casi!

Una KB di esempio

- ▶ Consideriamo il seguente problema

La legge americana afferma che per un cittadino è un crimine vendere armi ad una nazione ostile. Lo stato di Nono, un nemico dell'America, possiede dei missili, e gli sono stati venduti tutti dal Colonnello West, un americano

- ▶ Dimostreremo che West è un criminale!
- ▶ Prima di tutto, rappresentiamo i fatti sotto forma di clausole definite della logica del primo ordine

Una KB di esempio

- ▶ Per un americano è un crimine vendere armi ad una nazione ostile

Americano(x) \wedge Arma(y) \wedge Vende(x,y,z) \wedge Ostile(z) \Rightarrow Criminale(x)

Una KB di esempio

- ▶ Per un americano è un crimine vendere armi ad una nazione ostile

Americano(x) \wedge Arma(y) \wedge Vende(x,y,z) \wedge Ostile(z) \Rightarrow Criminale(x)

- ▶ Nono possiede dei missili, ovvero $\exists x$ Possiede(Nono, x) \wedge Missile(x):

Possiede(Nono, M_1) e Missile(M_1)

Una KB di esempio

- ▶ Per un americano è un crimine vendere armi ad una nazione ostile

Americano(x) \wedge Arma(y) \wedge Vende(x,y,z) \wedge Ostile(z) \Rightarrow Criminale(x)

- ▶ Nono possiede dei missili, ovvero $\exists x$ Possiede(Nono, x) \wedge Missile(x):

Possiede(Nono, M_1) e Missile(M_1)

- ▶ Tutti i missili sono stati venduti dal Colonnello West

Missile(x) \wedge Possiede(Nono, x) \Rightarrow Vende(West, x, Nono)

Una KB di esempio

- ▶ Per un americano è un crimine vendere armi ad una nazione ostile

Americano(x) \wedge Arma(y) \wedge Vende(x,y,z) \wedge Ostile(z) \Rightarrow Criminale(x)

- ▶ Nono possiede dei missili, ovvero $\exists x$ Possiede(Nono, x) \wedge Missile(x):

Possiede(Nono, M_1) e Missile(M_1)

- ▶ Tutti i missili sono stati venduti dal Colonnello West

Missile(x) \wedge Possiede(Nono, x) \Rightarrow Vende(West, x, Nono)

- ▶ I missili sono armi

Missile(x) \Rightarrow Arma(x)

- ▶ ...

Una KB di esempio

- ▶ *Un nemico dell'America viene considerato ostile*

Nemico(x, America) \Rightarrow Ostile(x)

Una KB di esempio

- ▶ *Un nemico dell'America viene considerato ostile*

Nemico(x, America) \Rightarrow Ostile(x)

- ▶ *West è americano*

Americano(West)

Una KB di esempio

- ▶ *Un nemico dell'America viene considerato ostile*
Nemico(x, America) \Rightarrow Ostile(x)
 - ▶ *West è americano*
Americano(West)
 - ▶ *Lo stato di Nono è un nemico dell'America*
Nemico(Nono, America)
-
- ▶ Questa KB non contiene simboli di funzione ed è un esempio di KB che prende il nome di **Datalog**
 - ▶ Rende l'inferenza molto più semplice

Una esempio di esecuzione

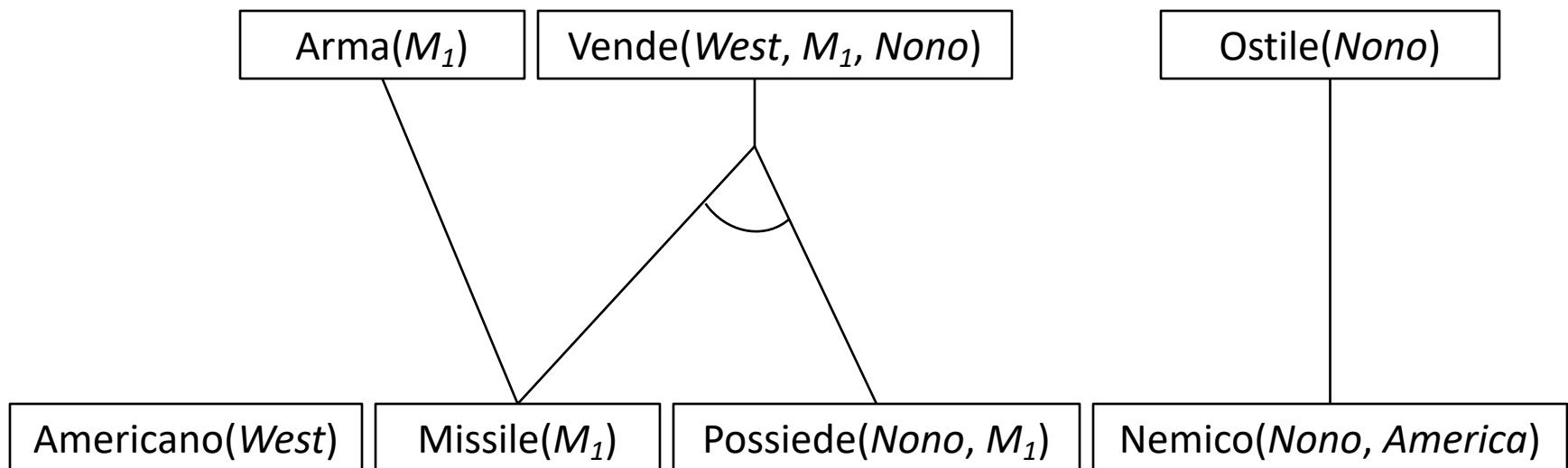
Americano(*West*)

Missile(M_1)

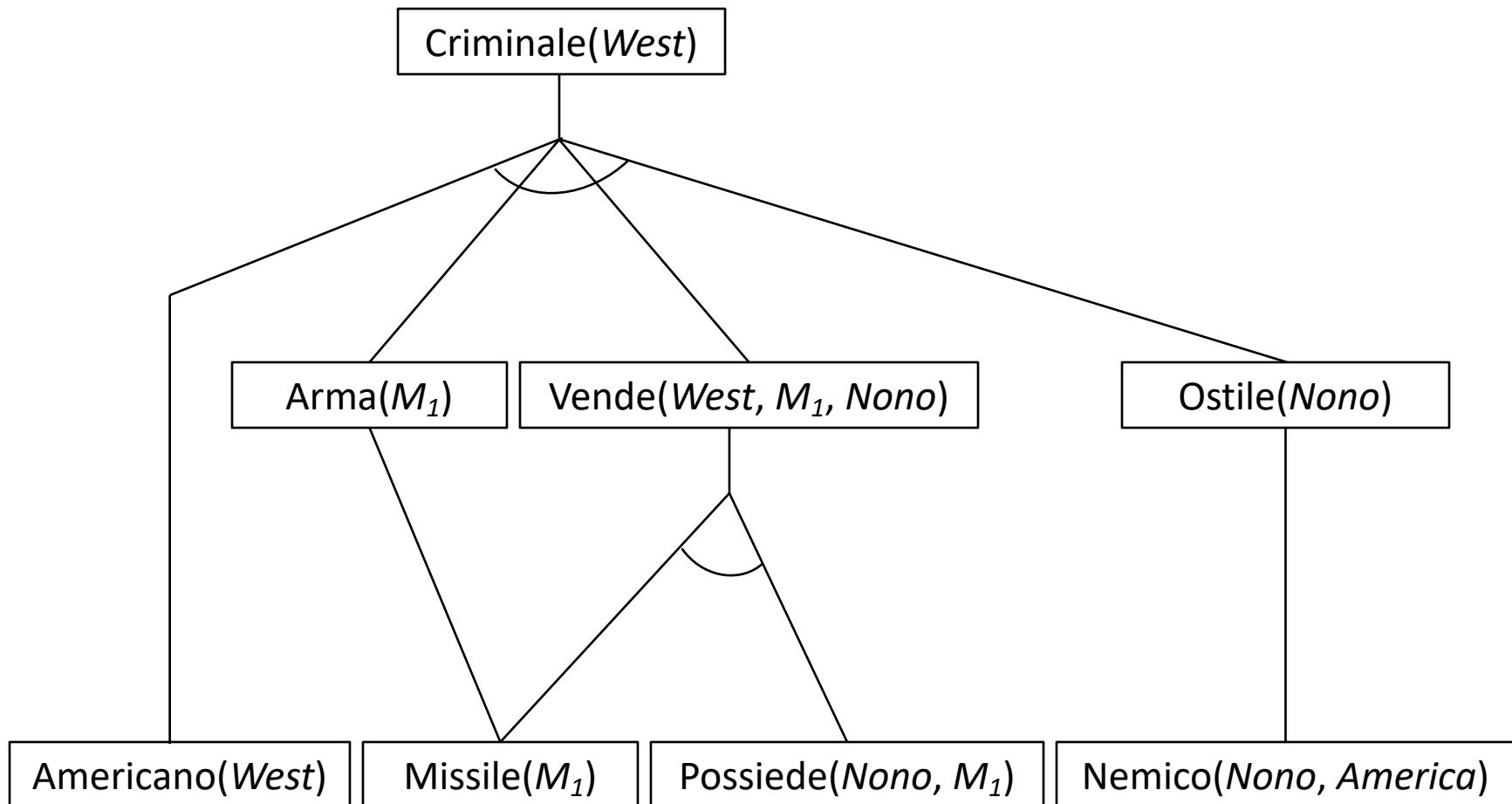
Possiede(*Nono*, M_1)

Nemico(*Nono*, *America*)

Una esempio di esecuzione



Una esempio di esecuzione



Proprietà del Forward Chaining

- ▶ L'algoritmo **FOL-CA-ASK** è corretto
 - ▶ Ogni inferenza è un'applicazione del Modus Ponens
- ▶ L'algoritmo **FOL-CA-ASK** è completo
 - ▶ La sua dimostrazione è simile a quella vista per la logica proposizionale
- ▶ Nel caso di KB Datalog (non contengono simboli di funzione) l'algoritmo termina in tempo polinomiale
 - ▶ Sia k la massima arità (numero di argomenti) tra tutti i predicati, p il numero di predicati ed n quello dei simboli costante
 - ▶ Non possono esserci più di $p n^k$ fatti distinti
- ▶ Tuttavia, come vale in generale, se α non è conseguenza logica di KB, allora l'algoritmo potrebbe non terminare!

Backward Chaining

- ▶ Abbiamo già visto un algoritmo di **backward chaining** (**concatenazione all'indietro**) per clausole definite proposizionali
- ▶ Si seguono le regole a ritroso per trovare fatti noti che supportino la dimostrazione
- ▶ Vedremo come si può applicare l'algoritmo a clausole definite del primo ordine
- ▶ Questo tipo di algoritmo è molto utilizzato nella programmazione logica
 - ▶ La forma più diffusa di ragionamento automatico

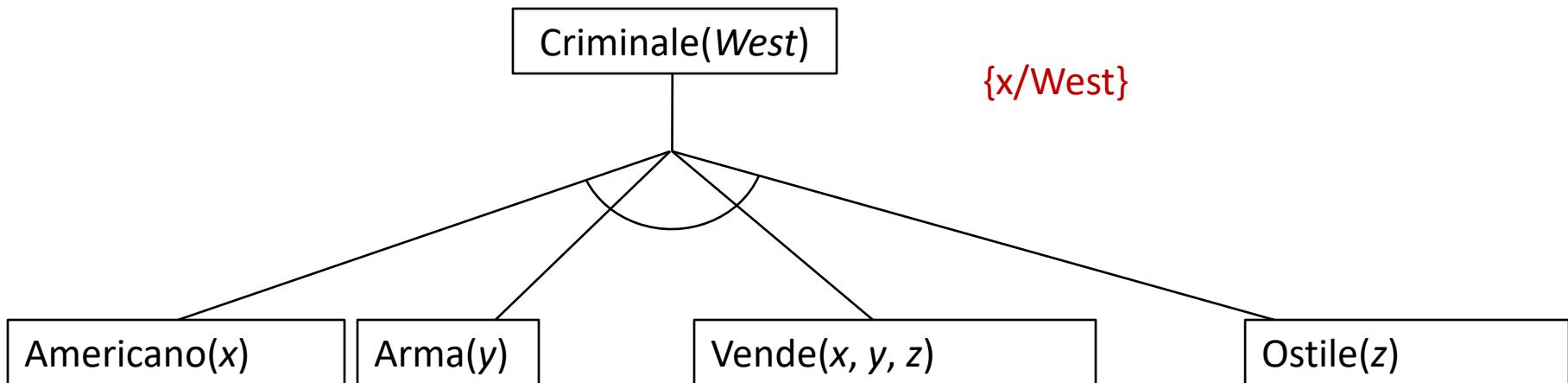
Backward Chaining: Algoritmo

- ▶ Il backward chaining è un tipo particolare di ricerca AND/OR
- ▶ La parte OR si deve al fatto che la query obiettivo può essere dimostrata da qualsiasi regola della KB (**FOL-CI-OR**)
 - ▶ Opera cercando tutte le clausole che potrebbero unificare con l'obiettivo
 - ▶ standardizzando le variabili nella clausola in modo che risultino variabili del tutto nuove
 - ▶ e verificando se la *rhs* della clausola unifica con l'obiettivo
- ▶ La parte AND si deve al fatto che tutti i congiunti della lista *lhs* di una clausola devono essere dimostrati (**FOL-CI-AND**)

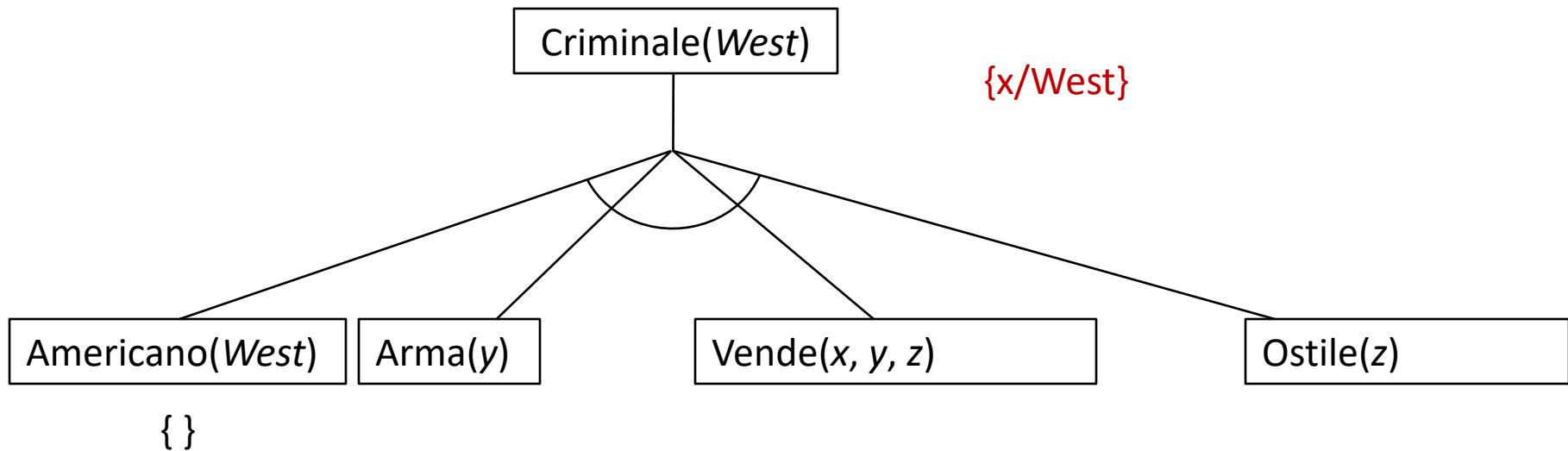
Una esempio di esecuzione

Criminale(*West*)

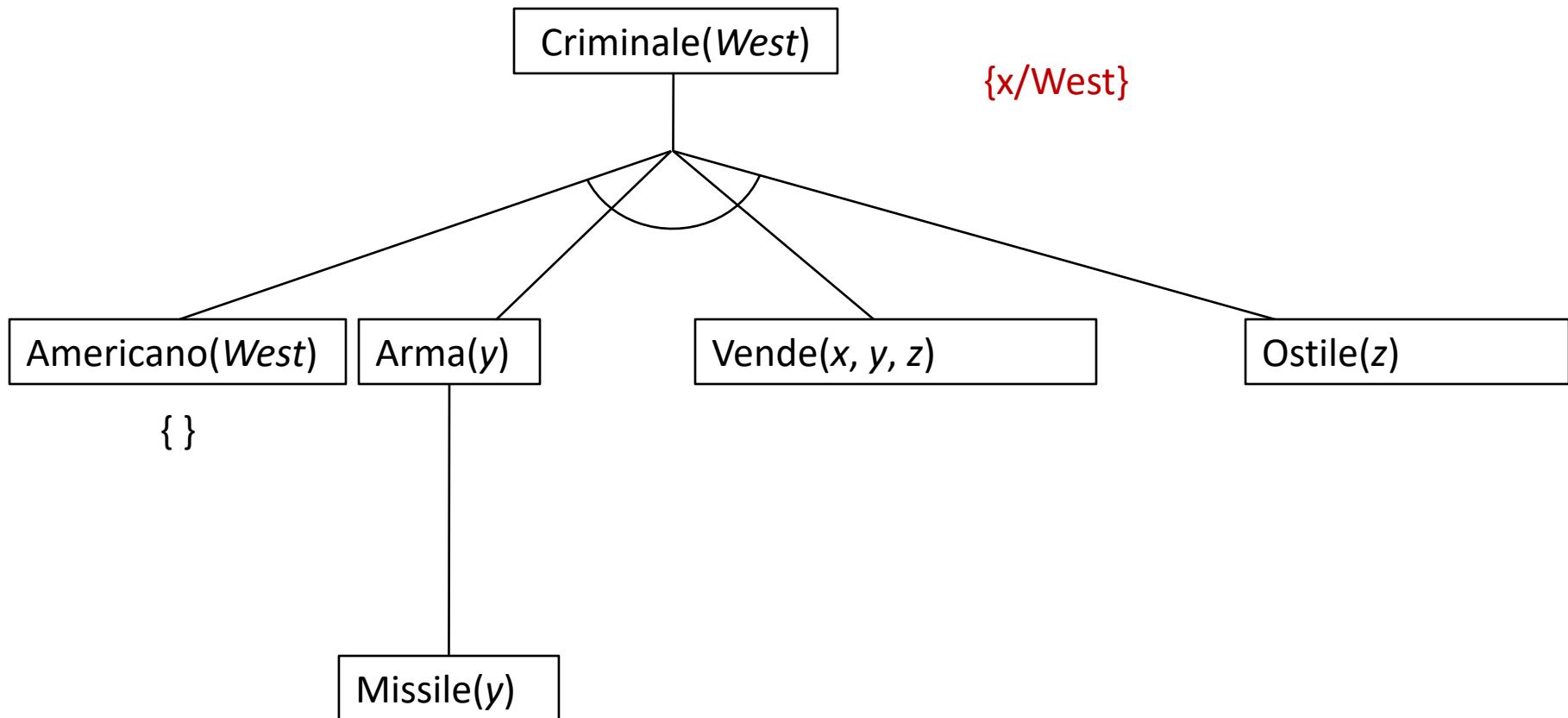
Una esempio di esecuzione



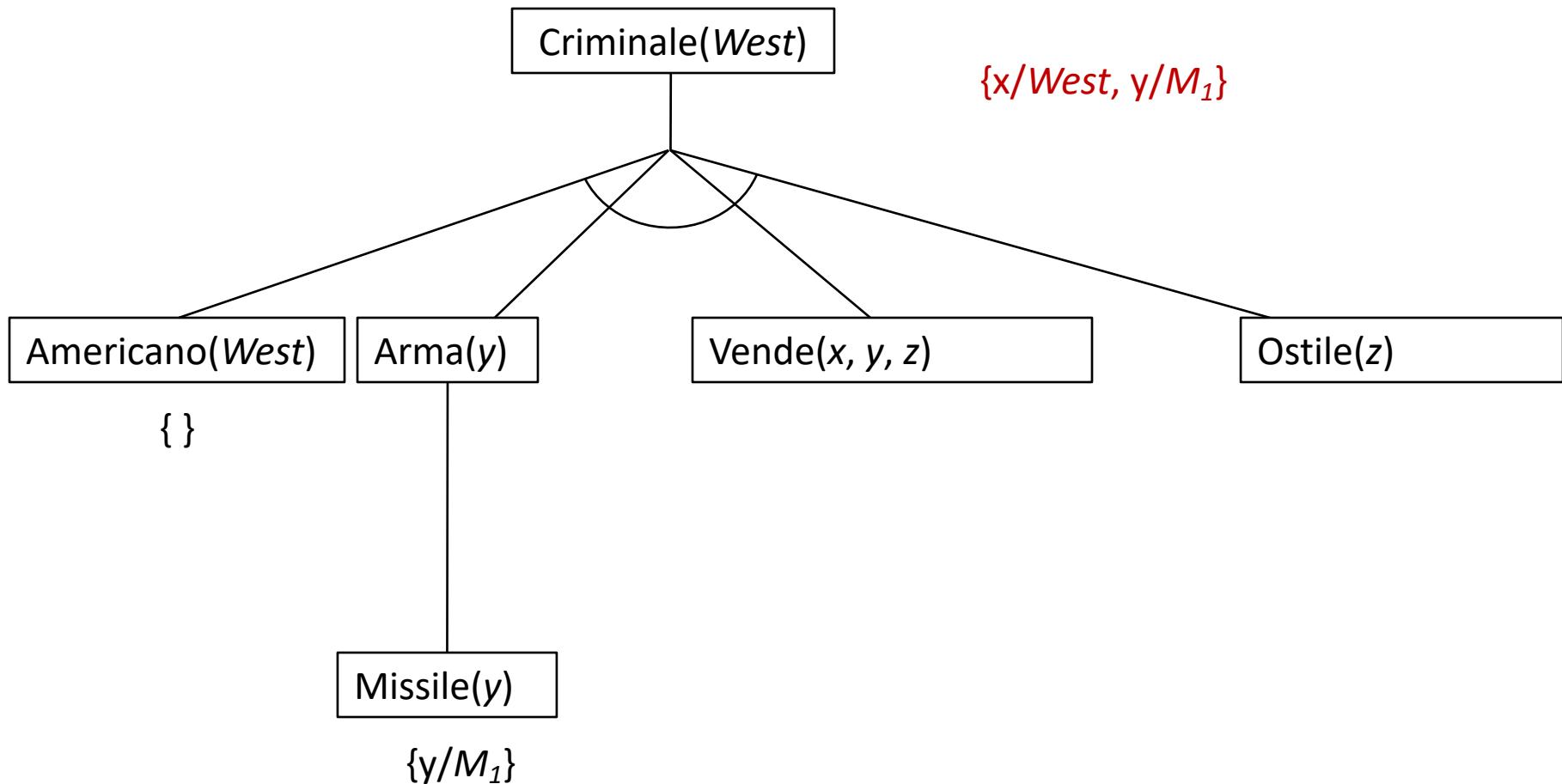
Una esempio di esecuzione



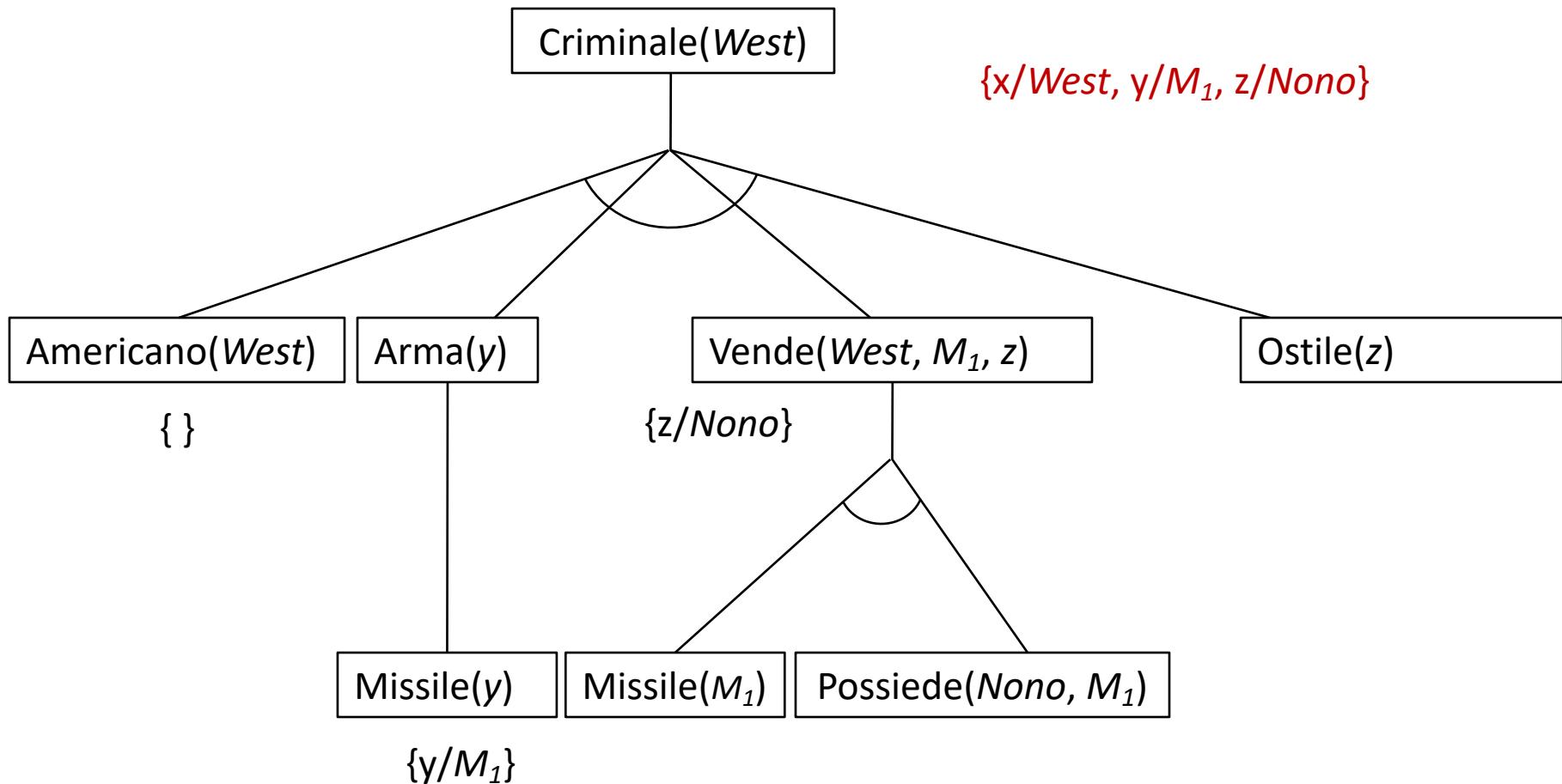
Una esempio di esecuzione



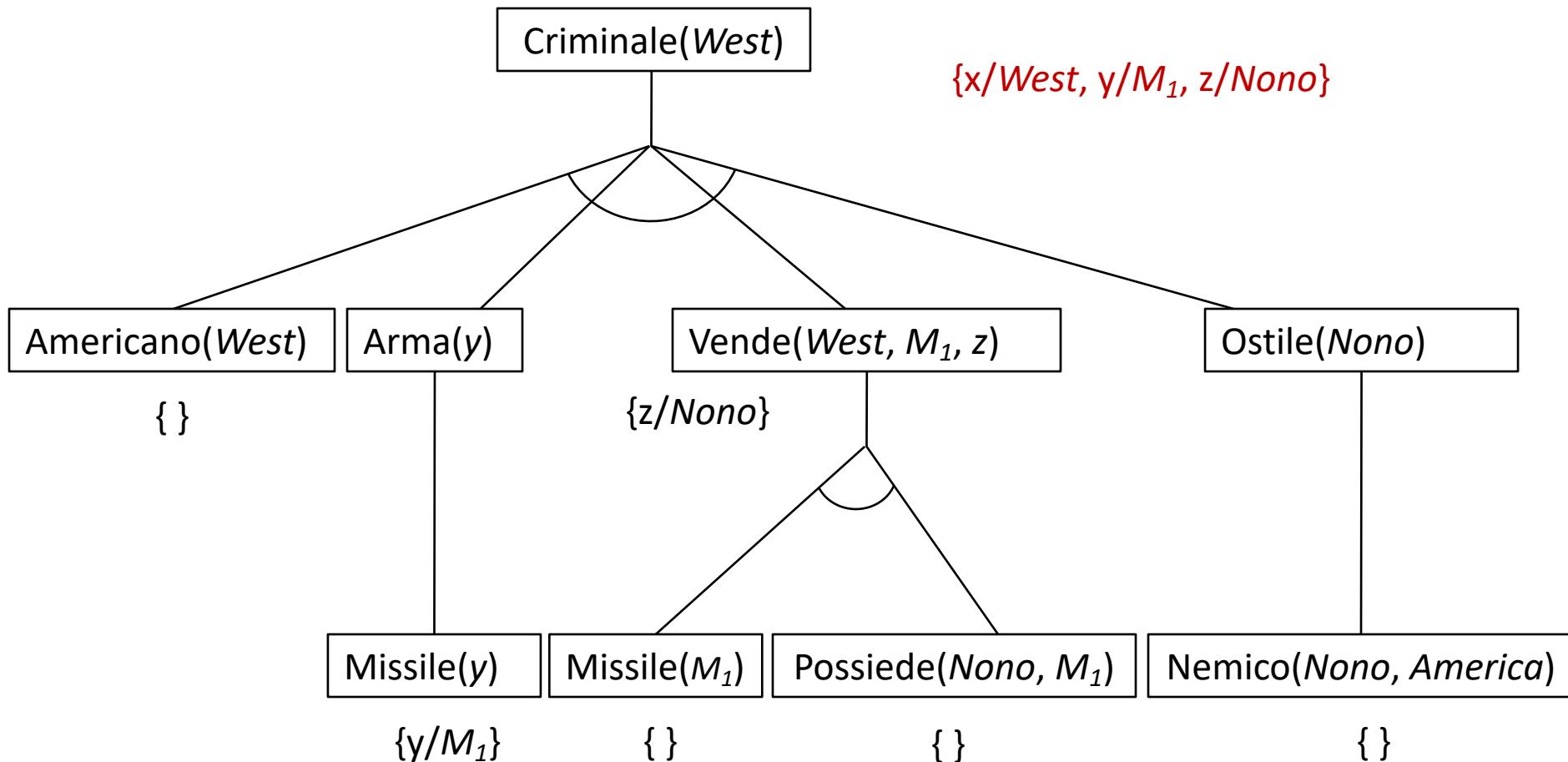
Una esempio di esecuzione



Una esempio di esecuzione



Una esempio di esecuzione



Proprietà del Backward Chaining

- ▶ La soluzione vista per il backward chaining è chiaramente un algoritmo di ricerca in profondità
- ▶ I suoi requisiti spaziali sono lineari nella dimensione della dimostrazione
- ▶ Incompletezza dovuta ai cicli infiniti
 - ▶ È possibile risolvere questo problema effettuando un controllo sul goal corrente versus gli altri goal memorizzati in uno stack
- ▶ Inefficienza dovuta alla ripetizione dei subgoal (sia in caso di successo che di fallimento)
 - ▶ È possibile risolvere questo problema effettuando caching dei risultati precedenti (**spazio extra!!!**)

Outline

- ▶ Inferenza del primo ordine
- ▶ Unificazione
- ▶ Modus Ponens generalizzato
- ▶ Forward/Backward chaining
- ▶ La programmazione logica
- ▶ Risoluzione

Risoluzione

- ▶ Abbiamo visto che la risoluzione è una procedura di inferenza completa per refutazione nella logica proposizionale
 - ▶ **Teorema di Deduzione:** $\text{KB} \models \alpha$ se e soltanto se $(\text{KB} \Rightarrow \alpha)$ è valida [cioè $\text{KB} \wedge \neg\alpha$ è insodd.]
 - ▶ **Estendiamo la sua applicazione alla logica del primo ordine**
- ▶ La regola di risoluzione per le clausole del primo ordine è una versione “sollevata” di quella proposizionale

$$\frac{l_1 \vee l_2 \vee \dots \vee l_k \quad m_1 \vee m_2 \vee \dots \vee m_n}{\text{SUBST}(\theta, l_1 \vee \dots \vee l_{i-1} \vee l_{i+1} \vee \dots \vee l_k \vee m_1 \vee \dots \vee m_{j-1} \vee m_{j+1} \vee \dots \vee m_n)}$$

dove $\text{UNIFY}(l_i, \neg m_j) = \theta$

Risoluzione

- ▶ Due clausole, che grazie alla standardizzazione separata presupponiamo sempre non avere alcuna variabile in comune, possono essere risolte se contengono letterali complementari
 - ▶ I letterali proposizionali sono complementari se uno è la negazione dell'altro; quelli di primo ordine lo sono se uno **unifica con** la negazione dell'altro
- ▶ Esempio
 - ▶ $[Animale(F(x)) \vee Ama(G(x), x)]$ e $[\neg Ama(u, v) \vee \neg Uccide(u, v)]$
 - ▶ $[Animale(F(x)) \vee \neg Uccide(G(x), x)]$ con unificatore $\theta = \{u/G(x), v/x\}$

Conversione in CNF

- ▶ È possibile, quindi, applicare passi di risoluzione a $\text{CNF}(\text{KB} \wedge \neg\alpha)$
- ▶ Questa metodologia di risoluzione risulta **completa** per KB in logica del primo ordine
- ▶ Vediamo, allora, come convertire le formule in forma normale congiuntiva (CNF)
 - ▶ Una congiunzione di clausole, ognuna delle quali è formata da una disgiunzione di letterali
 - ▶ I letterali possono contenere variabili, le quali sono sempre considerate universalmente quantificate

Conversione in CNF

- ▶ Esempio

- ▶ La formula

$$\forall x \text{ Americano}(x) \wedge \text{Arma}(y) \wedge \text{Vende}(x,y,z) \wedge \text{Ostile}(z) \Rightarrow \text{Criminale}(x)$$

- ▶ Diventa in CNF

$$\neg \text{Americano}(x) \vee \neg \text{Arma}(y) \vee \neg \text{Vende}(x,y,z) \vee \neg \text{Ostile}(z) \vee \text{Criminale}(x)$$

- ▶ La procedura per la conversione in CNF è molto simile a quella che abbiamo visto per il caso proposizionale
 - ▶ La differenza principale sorge dalla necessità di eliminare i quantificatori esistenziali

Conversione in CNF

- ▶ Illustriamo per passi la procedura generale, traducendo la seguente formula
 - ▶ *Chiunque ami tutti gli animali è amato da qualcuno*
 - ▶ $\forall x [\forall y \text{Animale}(y) \Rightarrow \text{Ama}(x,y)] \Rightarrow [\exists z \text{Ama}(z, x)]$
- ▶ I passi sono i seguenti
 - ▶ Eliminazione delle implicazioni
 - ▶ $\forall x [\neg\forall y \neg\text{Animale}(y) \vee \text{Ama}(x,y)] \vee [\exists z \text{Ama}(z, x)]$
 - ▶ Spostamento all'interno delle negazioni: $\neg\forall x p \equiv \exists x \neg p$ e $\neg\exists x p \equiv \forall x \neg p$
 - ▶ $\forall x [\exists y \neg(\neg\text{Animale}(y) \vee \text{Ama}(x,y))] \vee [\exists z \text{Ama}(z, x)]$
 - ▶ $\forall x [\exists y \neg\neg\text{Animale}(y) \wedge \neg\text{Ama}(x,y)] \vee [\exists z \text{Ama}(z, x)]$
 - ▶ $\forall x [\exists y \text{Animale}(y) \wedge \neg\text{Ama}(x,y)] \vee [\exists z \text{Ama}(z, x)]$

Conversione in CNF

- ▶ I passi sono i seguenti
 - ▶ Standardizzazione delle variabili: ogni quantificatore deve usare una variabile diversa
 - ▶ $\forall x [\exists y \text{ Animale}(y) \wedge \neg \text{Ama}(x, y)] \vee [\exists z \text{ Ama}(z, x)]$
 - ▶ Skolemizzazione: una forma più generale di istanziazione esistenziale. Ogni variabile esistenziale viene sostituita da una **funzione di Skolem** che include variabili universalmente quantificate
 - ▶ $\forall x [\text{Animale}(F(x)) \wedge \neg \text{Ama}(x, F(x))] \vee [\text{Ama}(G(x), x)]$

Conversione in CNF

- ▶ I passi sono i seguenti
 - ▶ Omissione di quantificatori universali
 - ▶ $[\text{Animale}(F(x)) \wedge \neg \text{Ama}(x, F(x))] \vee [\text{Ama}(G(x), x)]$
 - ▶ Distribuzione di \vee su \wedge
 - ▶ $[\text{Animale}(F(x)) \vee \text{Ama}(G(x), x)] \wedge [\neg \text{Ama}(x, F(x))] \vee \text{Ama}(G(x), x)$
- ▶ Ora la formula è in CNF ed è composta da due clausole

*Ogni formula della logica del primo ordine può essere convertita in una formula CNF **inferenzialmente equivalente***

Un esempio di dimostrazione

- ▶ La risoluzione dimostra che $\text{KB} \models \alpha$ provando che $\text{KB} \wedge \neg\alpha$ non è soddisfacibile
- ▶ Per fare questo deve derivare la clausola vuota
- ▶ Vediamo una dimostrazione sull'esempio del colonnello criminale
 - ▶ Regola generale in CNF:
$$\neg\text{Americano}(x) \vee \neg\text{Arma}(y) \vee \neg\text{Vende}(x,y,z) \vee \neg\text{Ostile}(z) \vee \text{Criminale}(x)$$
 - ▶ Obiettivo negato:
$$\neg\text{Criminale}(\text{West})$$

Un esempio di dimostrazione

$\neg\text{Americano}(x) \vee \neg\text{Arma}(y) \vee \neg\text{Vende}(x,y,z) \vee \neg\text{Ostile}(z) \vee \text{Criminale}(x)$

$\neg\text{Criminale}(\text{West})$

Un esempio di dimostrazione

$\neg\text{Americano}(x) \vee \neg\text{Arma}(y) \vee \neg\text{Vende}(x,y,z) \vee \neg\text{Ostile}(z) \vee \text{Criminale}(x)$

$\neg\text{Criminale}(\text{West})$

Americano(*West*)

$\neg\text{Americano}(\text{West}) \vee \neg\text{Arma}(y) \vee \neg\text{Vende}(\text{West},y,z) \vee \neg\text{Ostile}(z)$

Un esempio di dimostrazione

$\neg\text{Americano}(x) \vee \neg\text{Arma}(y) \vee \neg\text{Vende}(x,y,z) \vee \neg\text{Ostile}(z) \vee \text{Criminale}(x)$

$\neg\text{Criminale}(\text{West})$

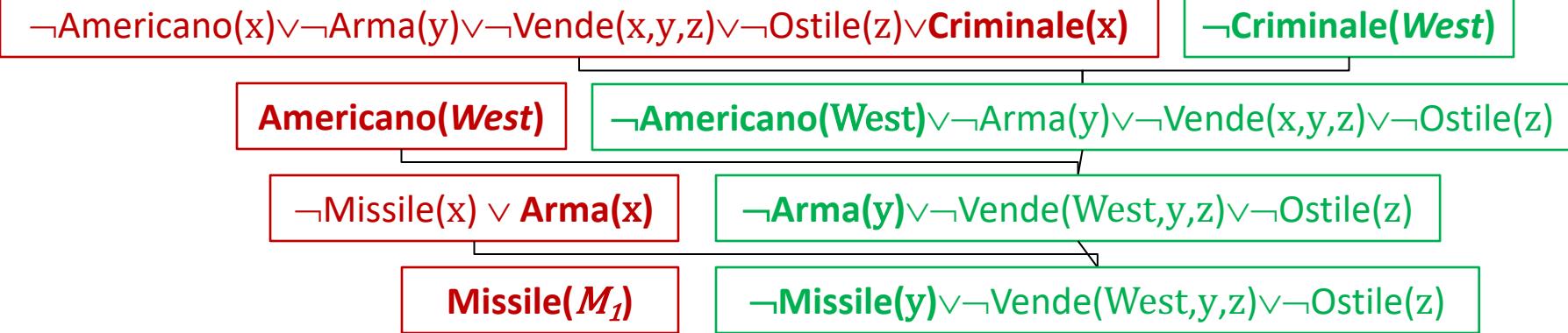
$\text{Americano}(\text{West})$

$\neg\text{Americano}(\text{West}) \vee \neg\text{Arma}(y) \vee \neg\text{Vende}(x,y,z) \vee \neg\text{Ostile}(z)$

$\neg\text{Missile}(x) \vee \text{Arma}(x)$

$\neg\text{Arma}(y) \vee \neg\text{Vende}(\text{West},y,z) \vee \neg\text{Ostile}(z)$

Un esempio di dimostrazione



Un esempio di dimostrazione

$\neg \text{Americano}(x) \vee \neg \text{Arma}(y) \vee \neg \text{Vende}(x,y,z) \vee \neg \text{Ostile}(z) \vee \text{Criminale}(x)$

$\neg \text{Criminale}(\text{West})$

$\text{Americano}(\text{West})$

$\neg \text{Americano}(\text{West}) \vee \neg \text{Arma}(y) \vee \neg \text{Vende}(x,y,z) \vee \neg \text{Ostile}(z)$

$\neg \text{Missile}(x) \vee \text{Arma}(x)$

$\neg \text{Arma}(y) \vee \neg \text{Vende}(\text{West},y,z) \vee \neg \text{Ostile}(z)$

$\neg \text{Missile}(M_1)$

$\neg \text{Missile}(y) \vee \neg \text{Vende}(\text{West},y,z) \vee \neg \text{Ostile}(z)$

$\neg \text{Missile}(x) \vee \neg \text{Possiede}(\text{Nono},x) \vee \text{Vende}(\text{West},x,\text{Nono})$

$\neg \text{Vende}(\text{West},M_1,z) \vee \neg \text{Ostile}(z)$

Un esempio di dimostrazione

$\neg \text{Americano}(x) \vee \neg \text{Arma}(y) \vee \neg \text{Vende}(x,y,z) \vee \neg \text{Ostile}(z) \vee \text{Criminale}(x)$

$\neg \text{Criminale}(\text{West})$

$\text{Americano}(\text{West})$

$\neg \text{Americano}(\text{West}) \vee \neg \text{Arma}(y) \vee \neg \text{Vende}(\text{West},y,z) \vee \neg \text{Ostile}(z)$

$\neg \text{Missile}(x) \vee \text{Arma}(x)$

$\neg \text{Arma}(y) \vee \neg \text{Vende}(\text{West},y,z) \vee \neg \text{Ostile}(z)$

$\text{Missile}(M_1)$

$\neg \text{Missile}(y) \vee \neg \text{Vende}(\text{West},y,z) \vee \neg \text{Ostile}(z)$

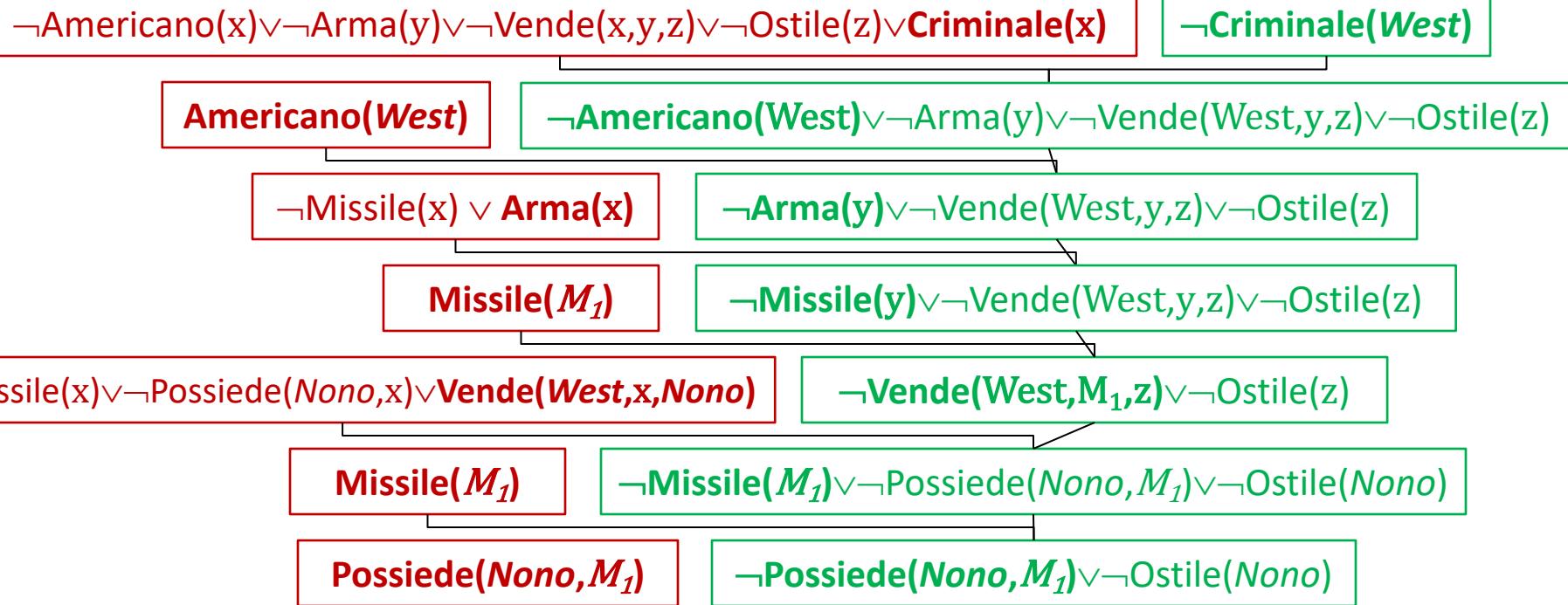
$\neg \text{Missile}(x) \vee \neg \text{Possiede}(\text{Nono},x) \vee \text{Vende}(\text{West},x,\text{Nono})$

$\neg \text{Vende}(\text{West},M_1,z) \vee \neg \text{Ostile}(z)$

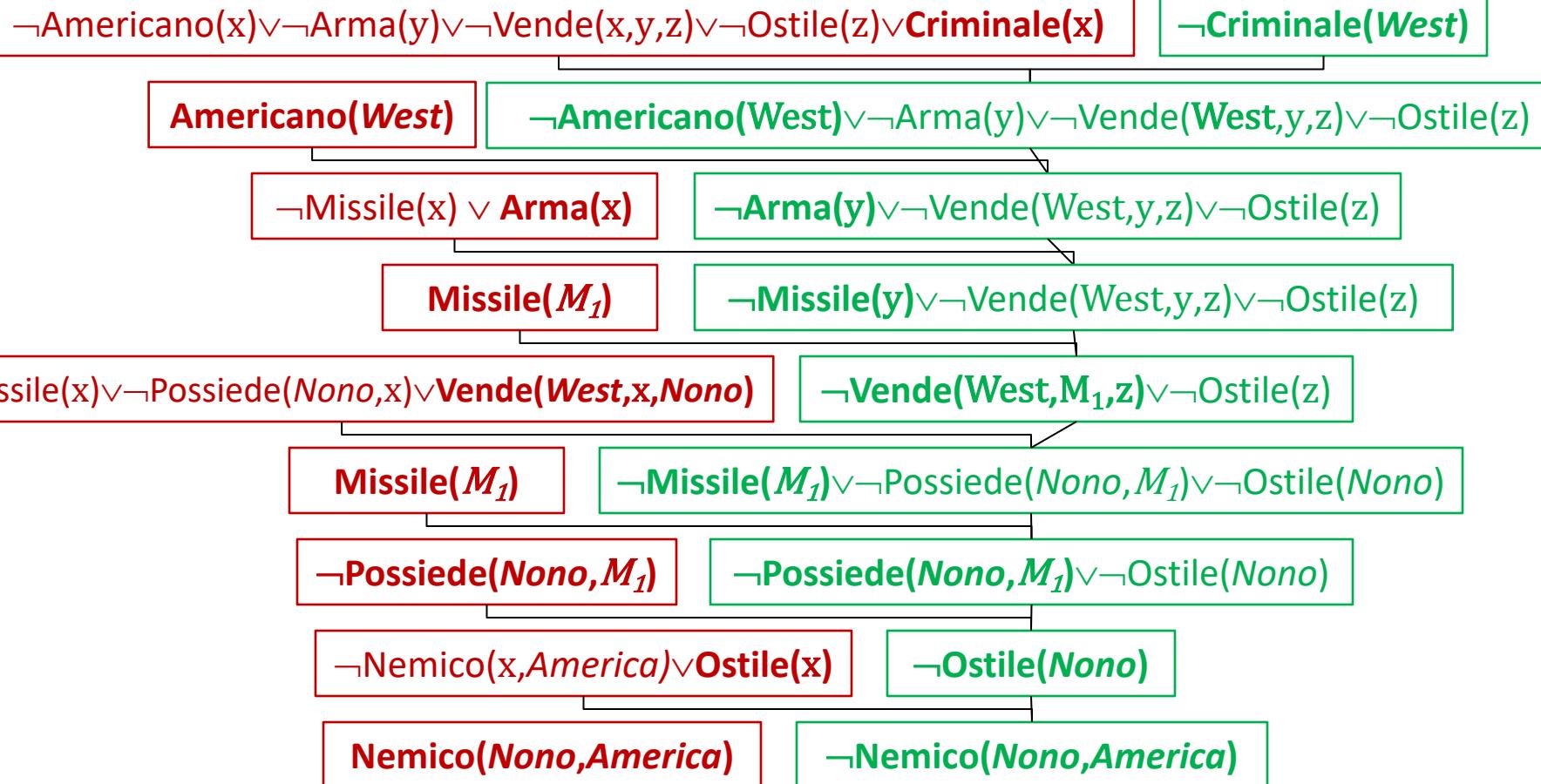
$\text{Missile}(M_1)$

$\neg \text{Missile}(M_1) \vee \neg \text{Possiede}(\text{Nono},M_1) \vee \neg \text{Ostile}(\text{Nono})$

Un esempio di dimostrazione



Un esempio di dimostrazione



Un esempio di dimostrazione

