

AGENTI LOGICI

CAPITOLO 7

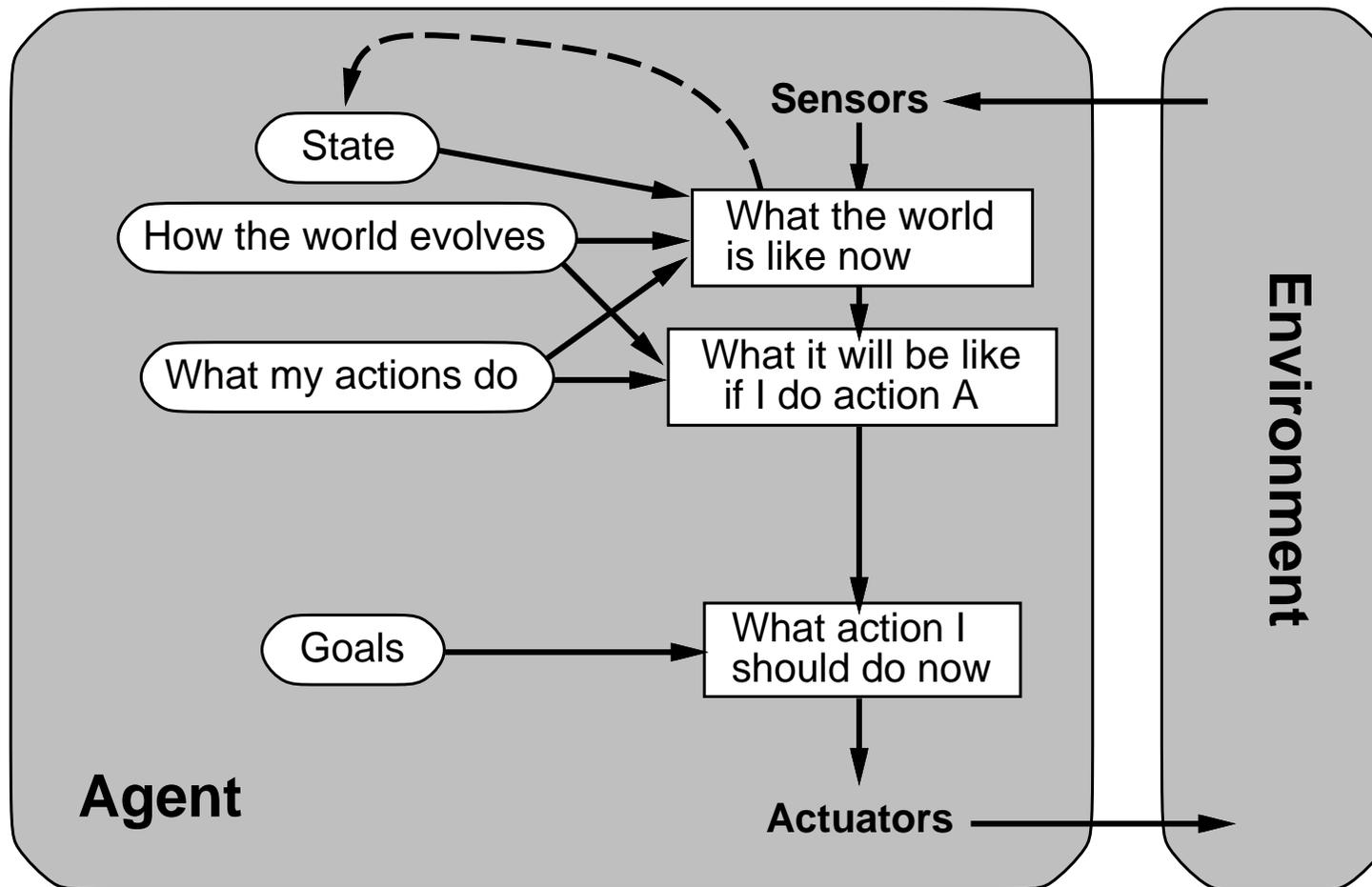
– docente: Alessandro Sperduti –

– presentazione basata sui lucidi di S. Russell –

Argomenti

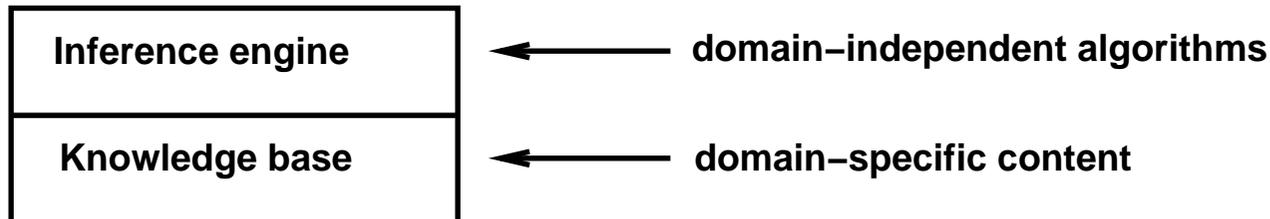
- ◇ Agenti basati sulla conoscenza
- ◇ Il mondo dei Wumpus
- ◇ Modelli ed entailment
- ◇ Regole di inferenza e dimostrazione di teoremi
 - forward chaining
 - backward chaining
 - risoluzione

Richiamo: Agente basato su goal



Vedremo che la logica fornisce un possibile formalismo per codificare la conoscenza necessaria (ovali) ad un agente (qualunque sia la sua architettura)

Base di conoscenza (Knowledge base)



Base di conoscenza = insieme di **sentenze** in un linguaggio **formale**

Approccio **dichiarativo** per la costruzione di un agente (o altro sistema):

DIRE (TELL) ad esso quello che ha bisogno di sapere

Quindi esso può CHIEDERE (ASK) a se stesso cosa fare—le risposte dovrebbero seguire dalla base di conoscenza (KB)

Gli agenti possono essere descritti al **livello della conoscenza**

cioè per quello che essi sanno, indipendentemente dall'implementazione

o a **livello implementativo**

cioè considerando le strutture dati nella KB e gli algoritmi che la manipolano

Un semplice agente basato sulla conoscenza

```
function KB-AGENTE(percezione) returns una azione  
  static: KB, una base di conoscenza  
           t, un contatore, inizializzato a 0, che indica il tempo  
  TELL(KB, COSTRUISCI-FORMULA-PERCEZIONE(percezione, t))  
  azione ← ASK(KB, COSTRUISCI-INTERROGAZIONE-AZIONE(t))  
  TELL(KB, COSTRUISCI-FORMULA-AZIONE(azione, t))  
  t ← t + 1  
  return azione
```

L'agente deve essere capace di:

Rappresentare stati, azioni, etc.

Incorporare nuove percezioni

Aggiornare le rappresentazioni interne del mondo (ambiente)

Dedurre proprietà nascoste del mondo

Dedurre le azioni appropriate da intraprendere

Il mondo dei Wumpus: descrizione PEAS

Misura di prestazione

oro +1000, morte -1000,

-1 per ogni spostamento, -10 per l'uso della freccia

Ambiente

Quadrati adiacenti ad un wumpus puzzano (stench)

Quadrati adiacenti ad una trappola (pit) sono ventilate

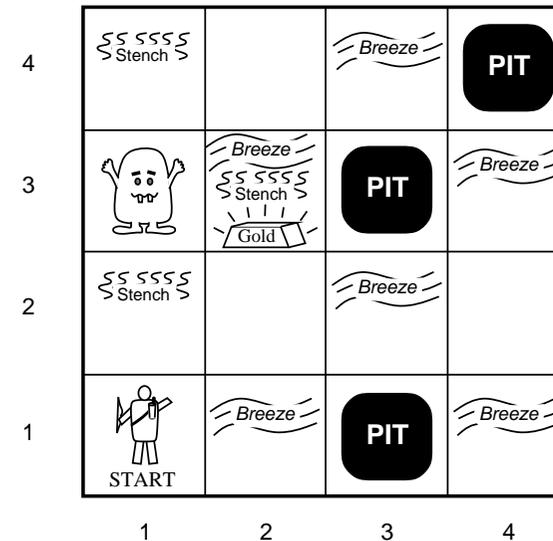
Luccichio (glitter) se e solo se l'oro è nello stesso quadrato

La freccia uccide il wumpus solo se l'agente è posto di fronte

La freccia può essere usata una sola volta

L'agente può prendere l'oro solo se si trova nello stesso quadrato

L'agente può lasciar cadere l'oro nel quadrato dove si trova



Sensori brezza, luccichio, puzza

Attuatori spostamento a sinistra, spostamento a destra,
avanti, prendi, lascia, lancia freccia

Caratterizzazione del mondo dei Wumpus

Osservabile??

Caratterizzazione del mondo dei Wumpus

Osservabile?? No—solo percezioni locali

Deterministico??

Caratterizzazione del mondo dei Wumpus

Osservabile?? No—solo percezioni **locali**

Deterministico?? Si—risultati delle azioni esattamente specificati

Episodico??

Caratterizzazione del mondo dei Wumpus

Osservabile?? No—solo percezioni **locali**

Deterministico?? Si—risultati delle azioni esattamente specificati

Episodico?? No—sequenziale al livello delle azioni

Statico??

Caratterizzazione del mondo dei Wumpus

Osservabile?? No—solo percezioni **locali**

Deterministico?? Si—risultati delle azioni esattamente specificati

Episodico?? No—sequenziale al livello delle azioni

Statico?? Si—Wumpus e trappole non si muovono

Discreto??

Caratterizzazione del mondo dei Wumpus

Osservabile?? No—solo percezioni **locali**

Deterministico?? Si—risultati delle azioni esattamente specificati

Episodico?? No—sequenziale al livello delle azioni

Statico?? Si—Wumpus e trappole non si muovono

Discreto?? Si

Agente Singolo??

Caratterizzazione del mondo dei Wumpus

Osservabile?? No—solo percezioni **locali**

Deterministico?? Si—risultati delle azioni esattamente specificati

Episodico?? No—sequenziale al livello delle azioni

Statico?? Si—Wumpus e trappole non si muovono

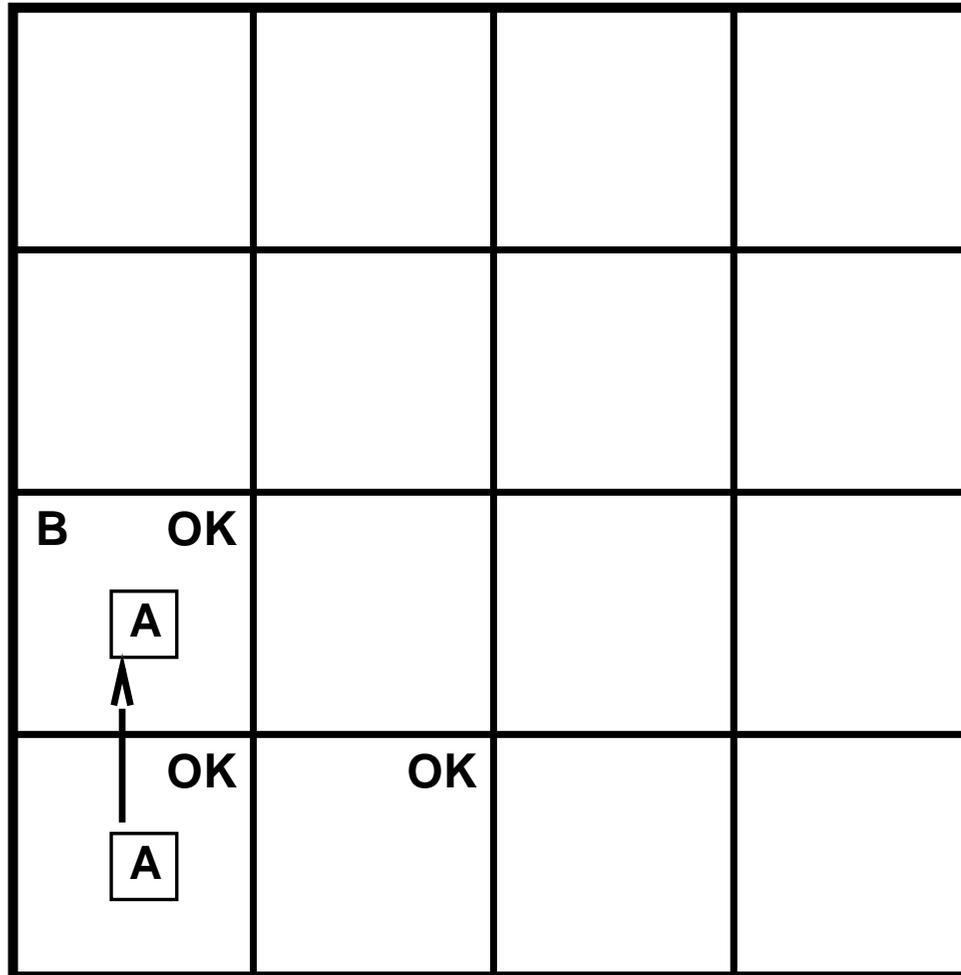
Discreto?? Si

Agente Singolo?? Si—Wumpus fa parte dell'ambiente

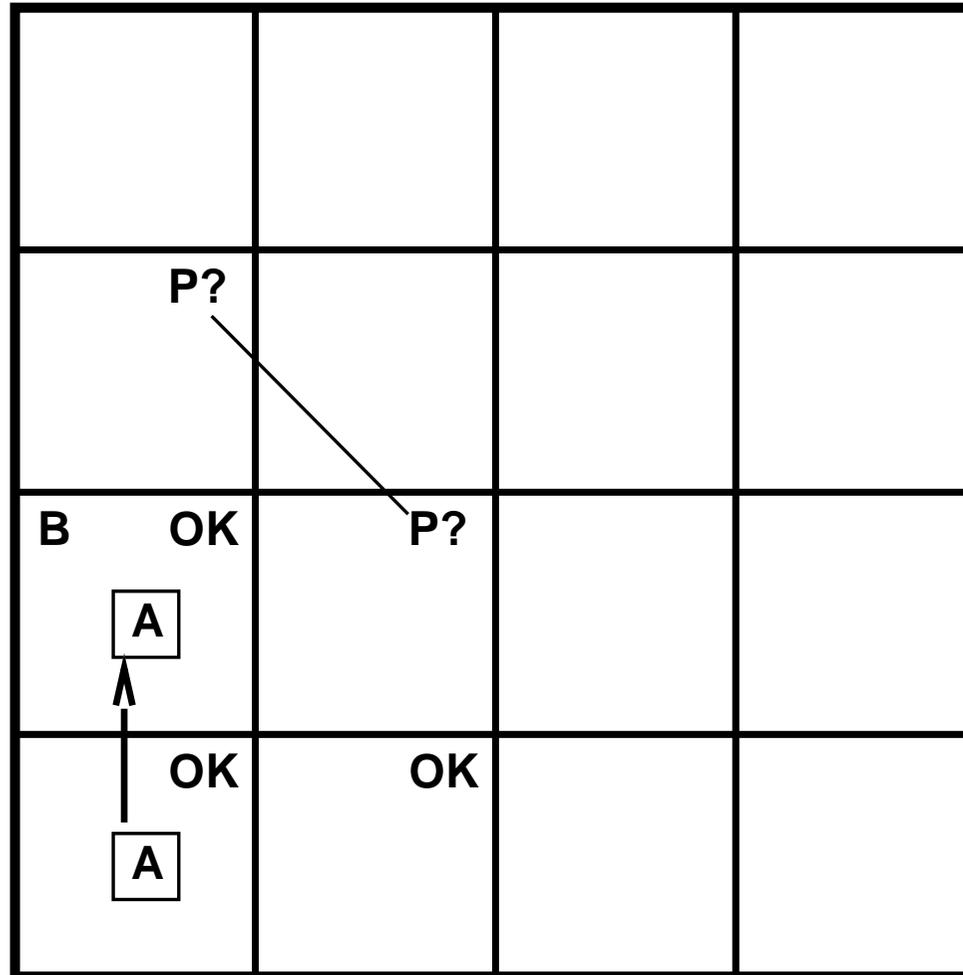
Esplorando il mondo dei wumpus

OK			
OK A	OK		

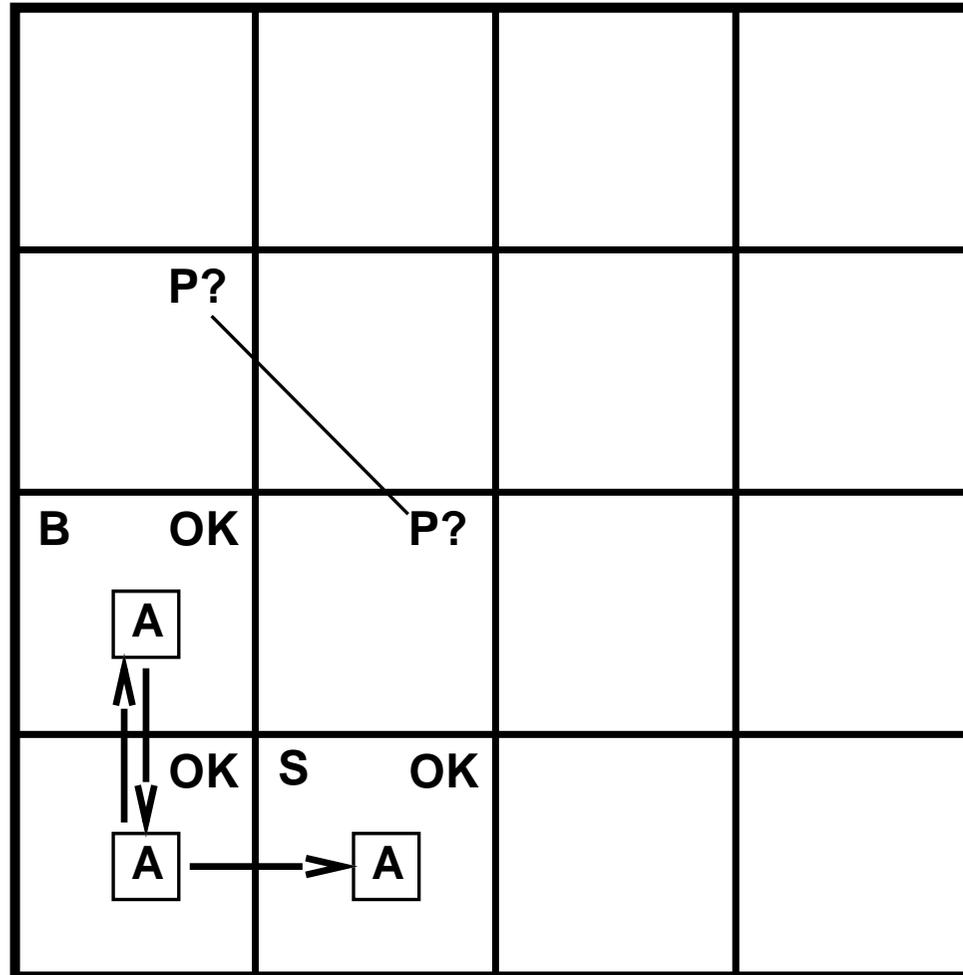
Esplorando il mondo dei wumpus



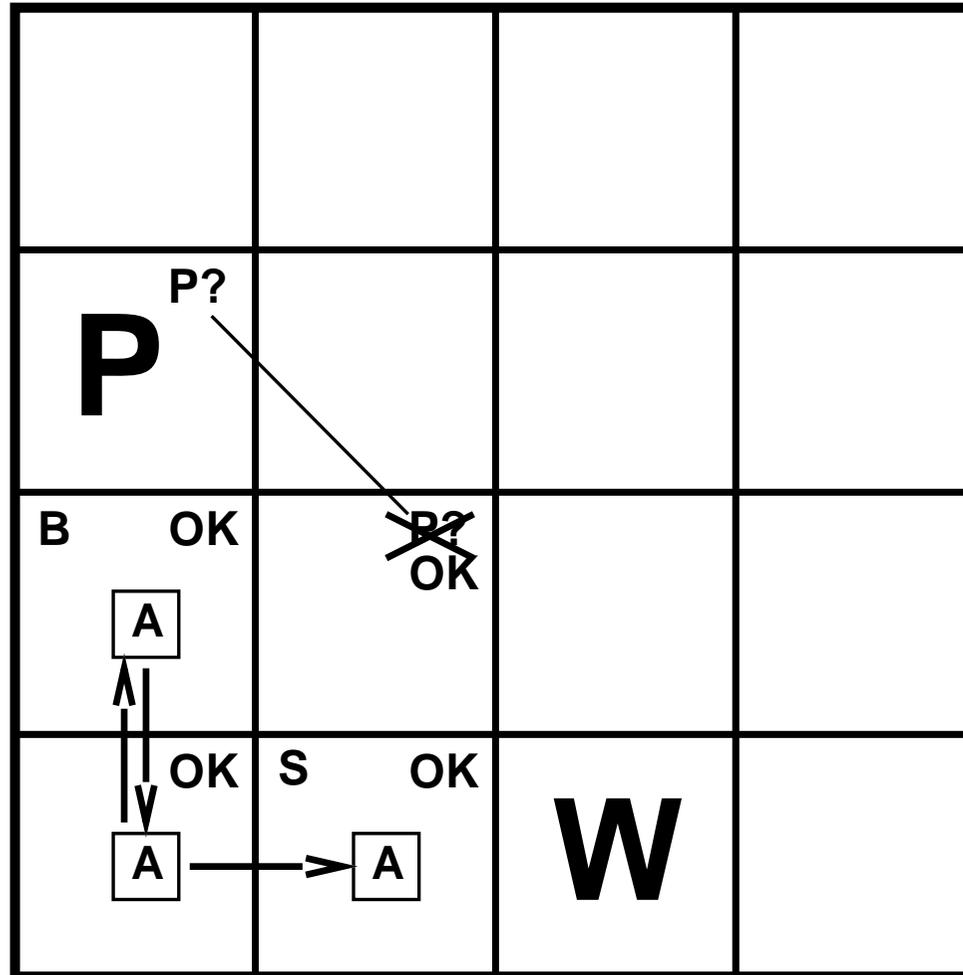
Esplorando il mondo dei wumpus



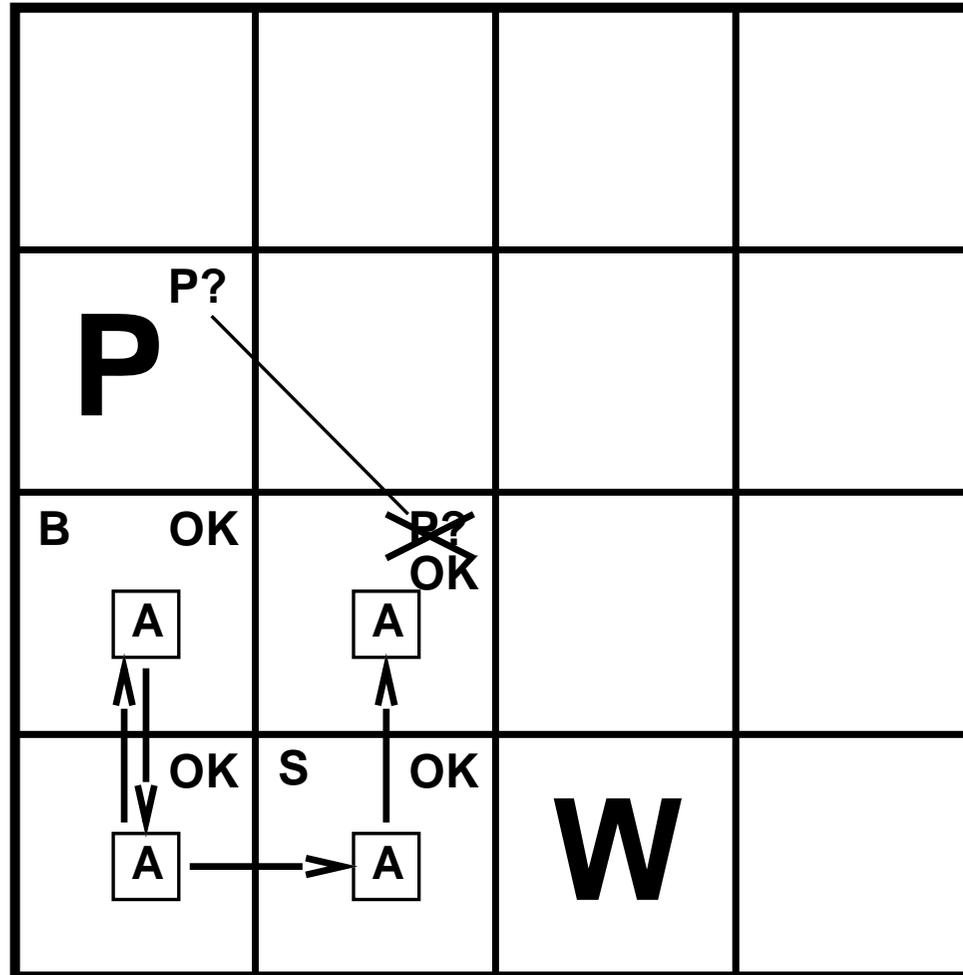
Esplorando il mondo dei wumpus



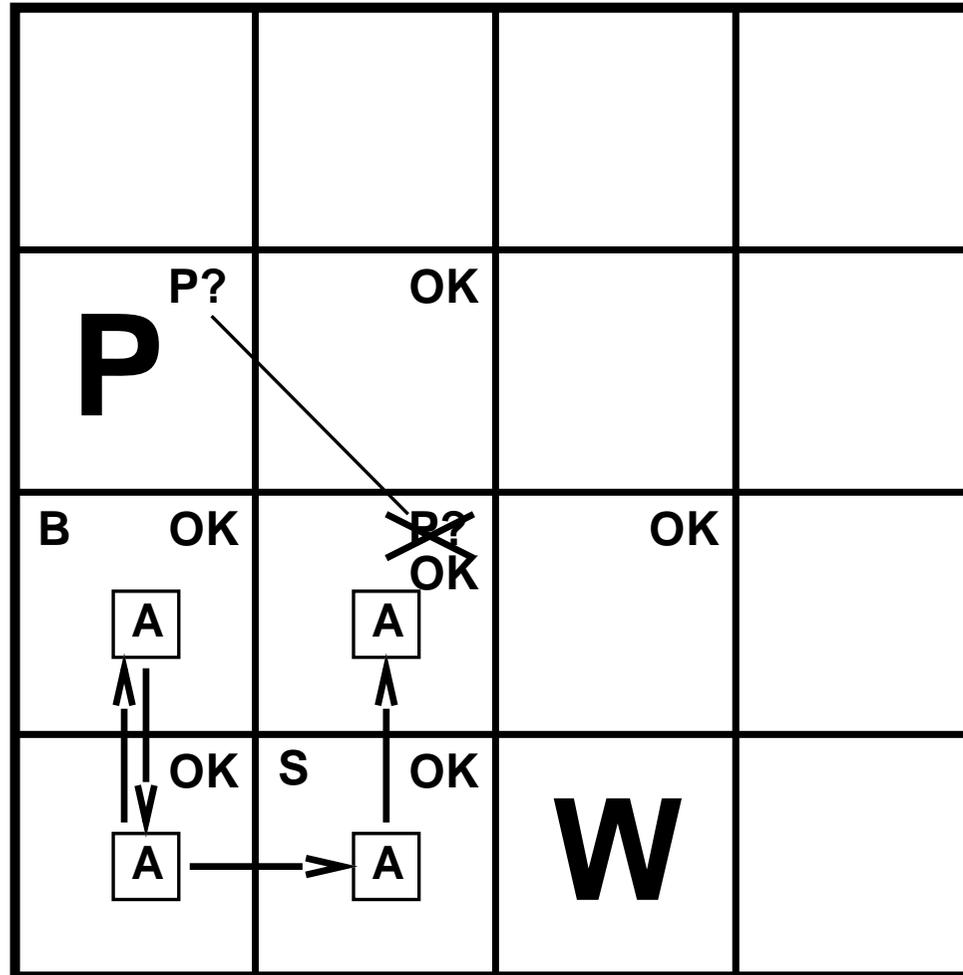
Esplorando il mondo dei wumpus



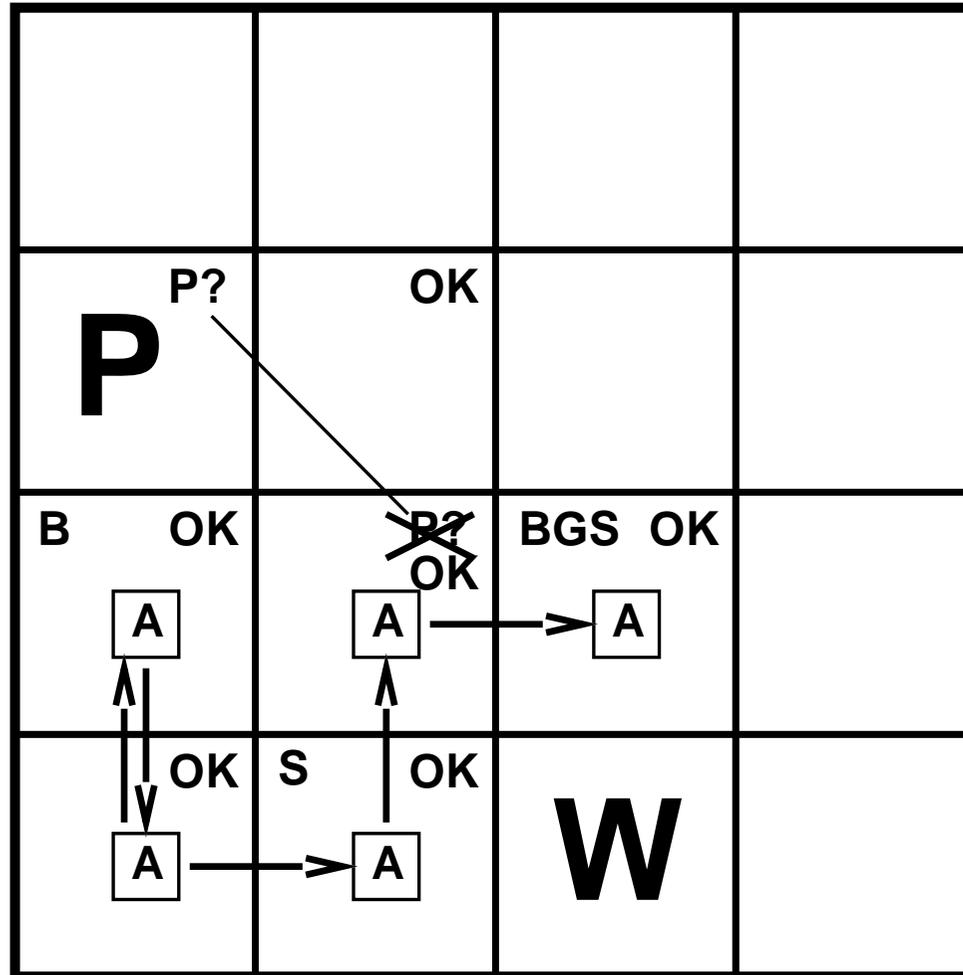
Esplorando il mondo dei wumpus



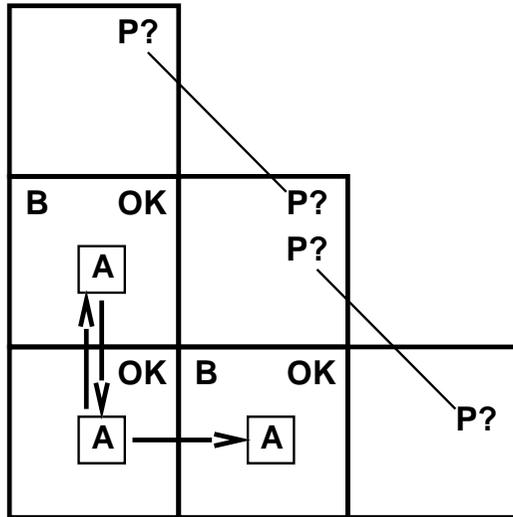
Esplorando il mondo dei wumpus



Esplorando il mondo dei wumpus



Altre situazioni critiche



Brezza in (1,2) e (2,1)

⇒ azioni non sicure

Assumendo le trappole uniformemente distribuite,

(2,2) contiene una trappola con prob 0.86, contro 0.31 per (1,2) e (2,1)

Puzza in (1,1)

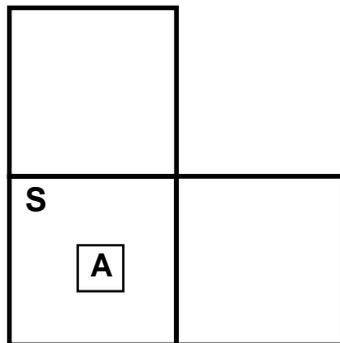
⇒ non si può muovere

Si può usare una strategia di **coercizione**:

lancia la freccia di fronte

wumpus presente ⇒ morto ⇒ sicuro

wumpus assente ⇒ sicuro



Modelli

I logici tipicamente pensano in termini di **modelli**, che formalmente sono mondi strutturati rispetto ai quali si può valutare la verità

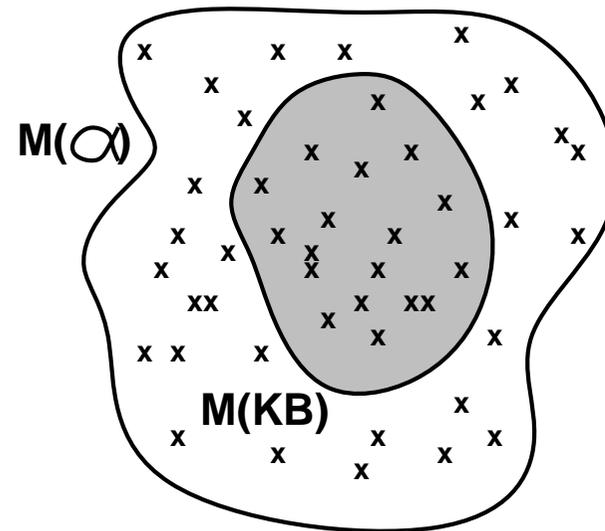
Diciamo che m è un **modello** di una sentenza α se α è vera in m

$M(\alpha)$ è l'insieme di tutti i modelli di α

Allora $KB \models \alpha$ se e solo se $M(KB) \subseteq M(\alpha)$

Per esempio, $KB = \text{l'INTER ha vinto e il MILAN ha vinto}$

$\alpha = \text{l'INTER ha vinto}$

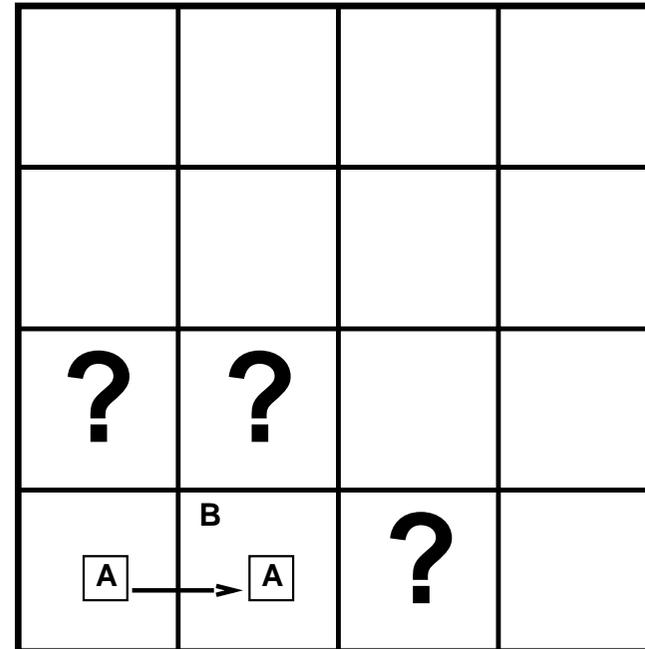


Entailment nel mondo dei wumpus

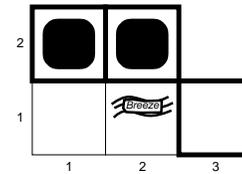
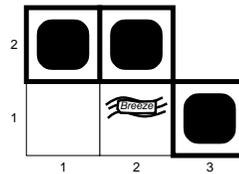
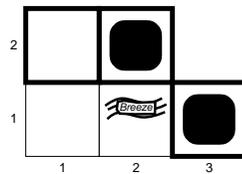
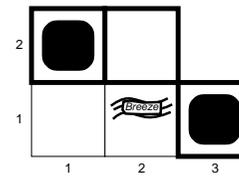
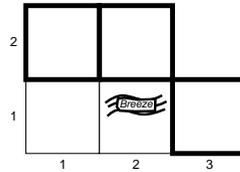
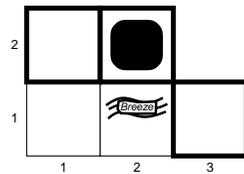
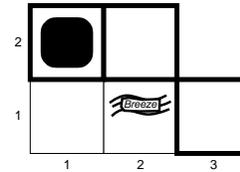
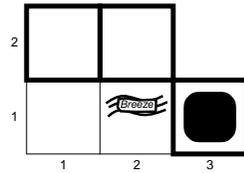
Situazione dopo aver riconosciuto che non c'è nulla in [1,1], spostamento a destra, brezza in [2,1]

Consideriamo modelli possibili per i "?" assumendo solo trappole

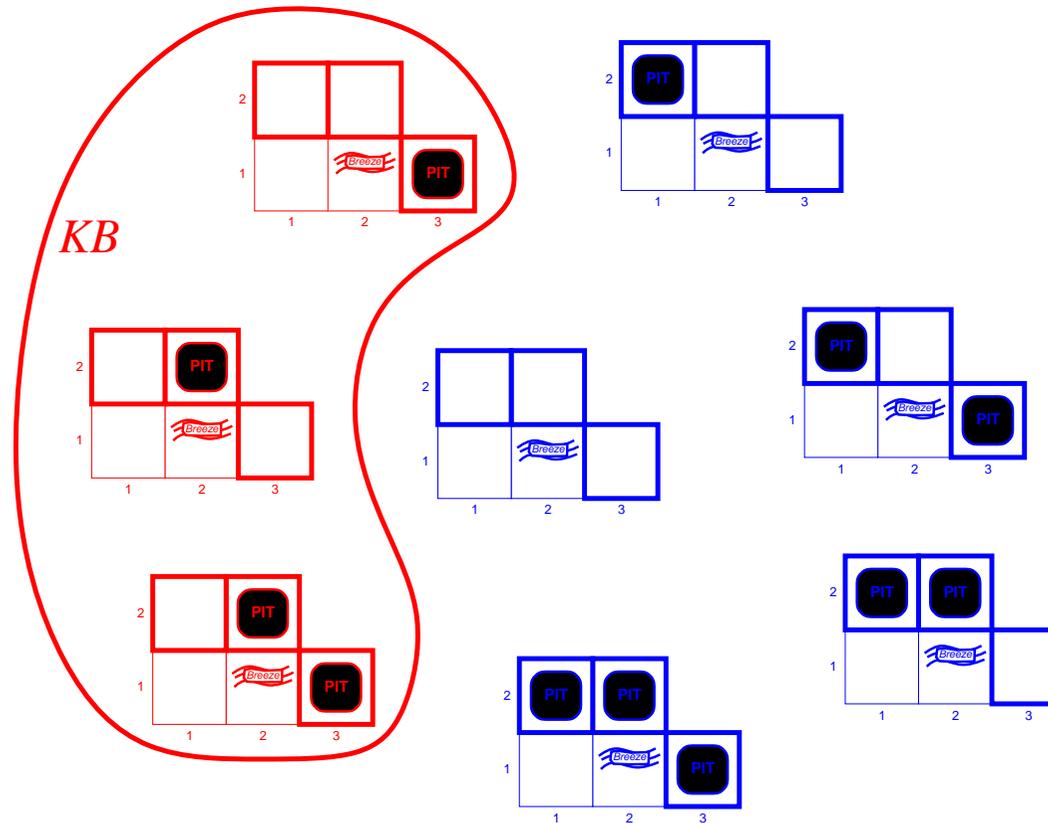
3 scelte Booleane \Rightarrow 8 possibili modelli



Modelli per il mondo dei wumpus

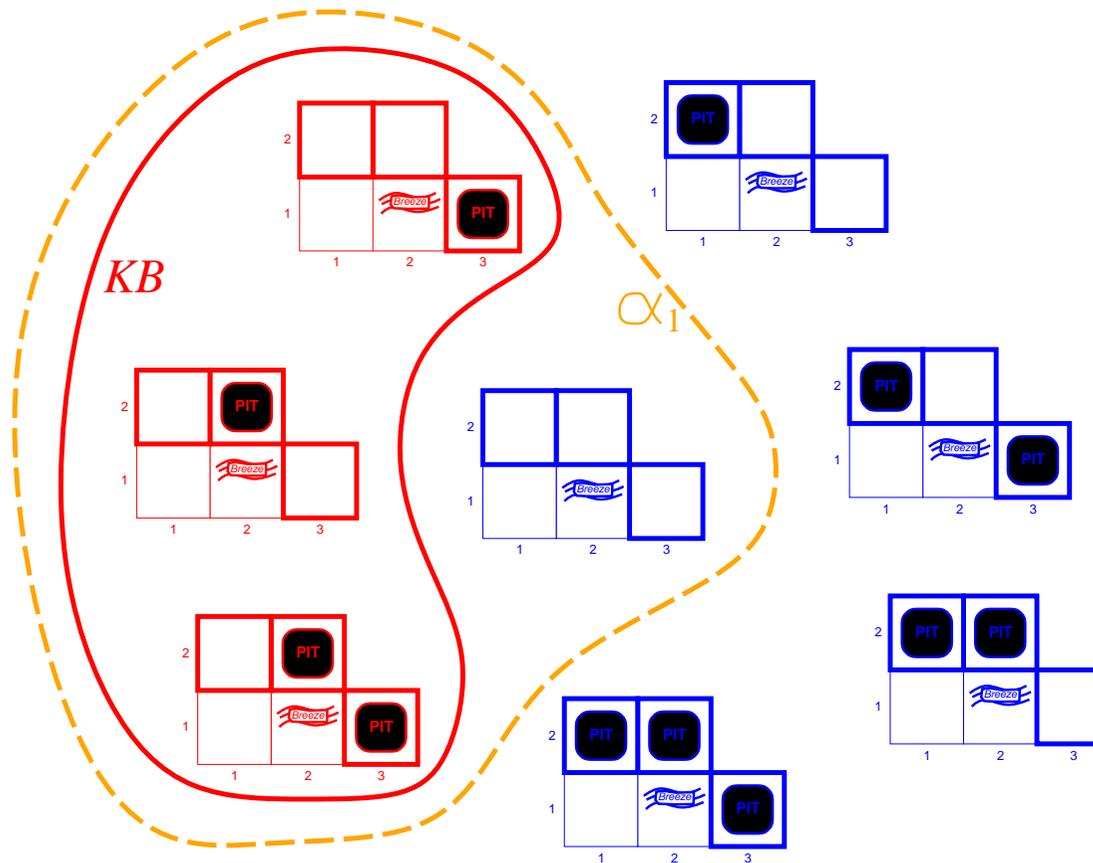


Modelli per il mondo dei wumpus



$KB = \text{regole del mondo dei wumpus} + \text{osservazioni}$

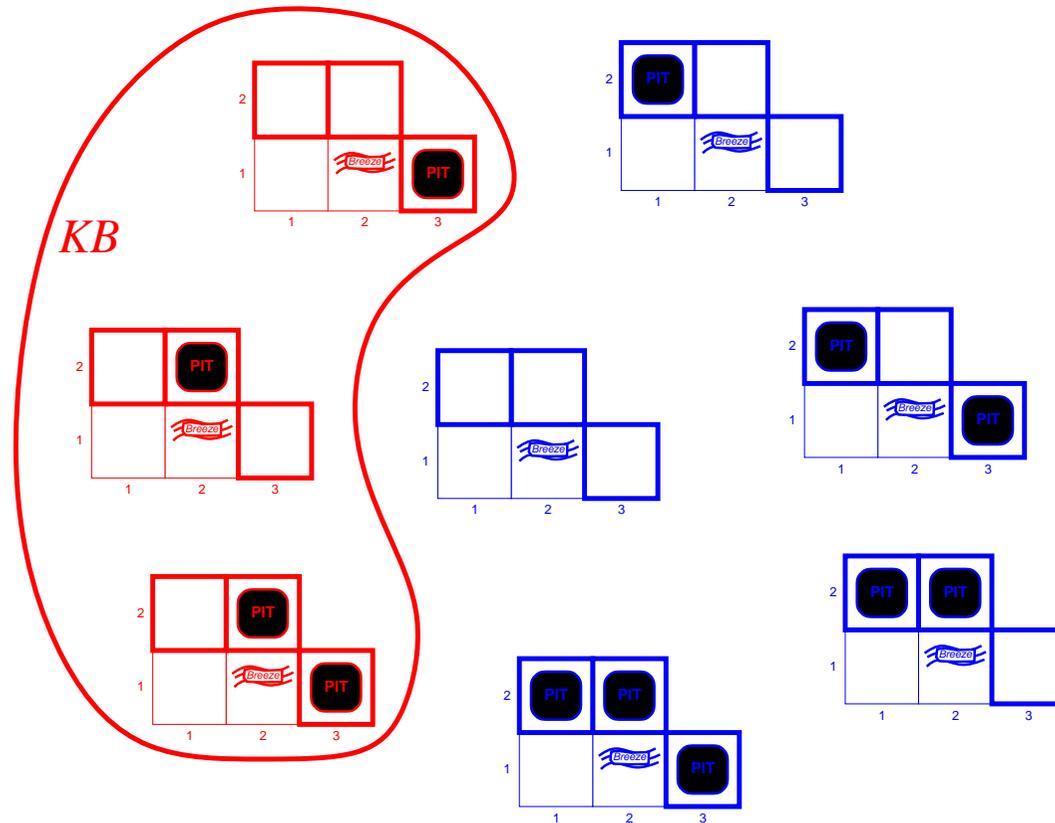
Modelli per il mondo dei wumpus



KB = regole del mondo dei wumpus + osservazioni

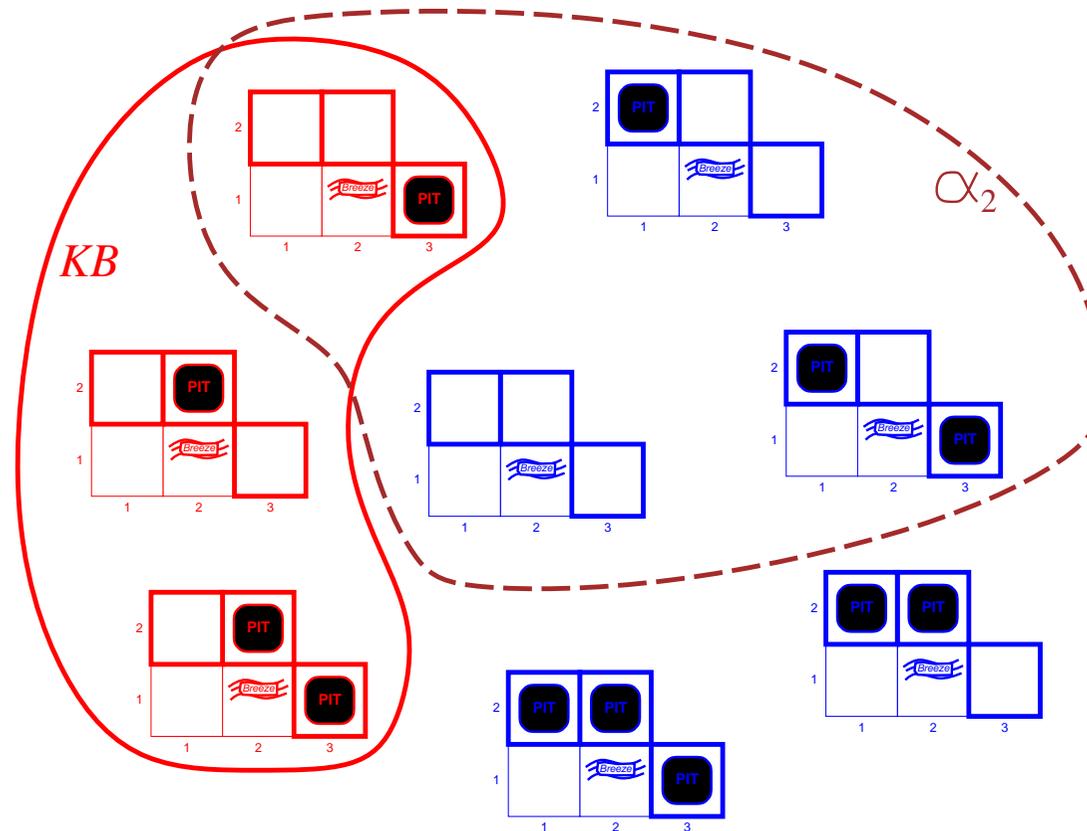
$\alpha_1 = "[1,2]$ è sicuro", $KB \models \alpha_1$, provato via model checking (test sui modelli)

Modelli per il mondo dei wumpus



$KB = \text{regole del mondo dei wumpus} + \text{osservazioni}$

Modelli per il mondo dei wumpus



$KB = \text{regole del mondo dei wumpus} + \text{osservazioni}$

$\alpha_2 = \text{“}[2,2] \text{ è sicuro”}, KB \not\models \alpha_2$

Sentenze nel mondo dei wumpus

Poniamo $P_{i,j}$ essere vero se c'è una trappola in $[i, j]$.

Poniamo $B_{i,j}$ essere vero se c'è brezza in $[i, j]$.

$$\neg P_{1,1}$$

$$\neg B_{1,1}$$

$$B_{2,1}$$

“Le trappole causano brezze in quadrati adiacenti”

Sentenze nel mondo dei wumpus

Poniamo $P_{i,j}$ essere vero se c'è una trappola in $[i, j]$.

Poniamo $B_{i,j}$ essere vero se c'è brezza in $[i, j]$.

$$\neg P_{1,1}$$

$$\neg B_{1,1}$$

$$B_{2,1}$$

“Le trappole causano brezze in quadrati adiacenti”

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

$$B_{2,1} \Leftrightarrow (P_{1,1} \vee P_{2,2} \vee P_{3,1})$$

“Un quadrato ha brezza *se e solo se* c'è una trappola adiacente”

Tabelle di verità per l'inferenza

$B_{1,1}$	$B_{2,1}$	$P_{1,1}$	$P_{1,2}$	$P_{2,1}$	$P_{2,2}$	$P_{3,1}$	KB	α_1
<i>falso</i>	<i>vero</i>							
<i>falso</i>	<i>falso</i>	<i>falso</i>	<i>falso</i>	<i>falso</i>	<i>falso</i>	<i>vero</i>	<i>falso</i>	<i>vero</i>
\vdots	\vdots							
<i>falso</i>	<i>vero</i>	<i>falso</i>	<i>falso</i>	<i>falso</i>	<i>falso</i>	<i>falso</i>	<i>falso</i>	<i>vero</i>
<i>falso</i>	<i>vero</i>	<i>falso</i>	<i>falso</i>	<i>falso</i>	<i>falso</i>	<i>vero</i>	<u><i>vero</i></u>	<u><i>vero</i></u>
<i>falso</i>	<i>vero</i>	<i>falso</i>	<i>falso</i>	<i>falso</i>	<i>vero</i>	<i>falso</i>	<u><i>vero</i></u>	<u><i>vero</i></u>
<i>falso</i>	<i>vero</i>	<i>falso</i>	<i>falso</i>	<i>falso</i>	<i>vero</i>	<i>vero</i>	<u><i>vero</i></u>	<u><i>vero</i></u>
<i>falso</i>	<i>vero</i>	<i>falso</i>	<i>falso</i>	<i>vero</i>	<i>falso</i>	<i>falso</i>	<i>falso</i>	<i>vero</i>
\vdots	\vdots							
<i>vero</i>	<i>falso</i>	<i>falso</i>						

Inferenza per mezzo di enumerazione

Una enumerazione a scandaglio (Depth-first) di tutti i modelli è corretta e completa

```
function TV-IMPLICA?(KB,  $\alpha$ ) returns true oppure false
```

```
  s  $\leftarrow$  una lista di simboli proposizionali contenuti in KB e  $\alpha$ 
```

```
  return TV-VERIFICA-TUTTO(KB,  $\alpha$ , s, [])
```

```
function TV-VERIFICA-TUTTO(KB,  $\alpha$ , s, modello) returns true oppure false
```

```
  if VUOTO?(s) then
```

```
    if CP-VERO?(KB, modello) then return CP-VERO?( $\alpha$ , modello)
```

```
    else return true
```

```
  else do
```

```
    P  $\leftarrow$  PRIMO(s); resto  $\leftarrow$  RESTO(s)
```

```
    return TV-VERIFICA-TUTTO(KB,  $\alpha$ , resto, ESTENDI(P, true, modello))
```

```
  and
```

```
    TV-VERIFICA-TUTTO(KB,  $\alpha$ , resto, ESTENDI(P, false, modello))
```

$O(2^n)$ per n simboli; il problema è co-NP-complete

Metodi di prova

I metodi di prova si suddividono in (approssimativamente) due tipologie:

Applicazione di regole di inferenza

- Generazione (corretta) di nuove sentenze a partire da vecchie sentenze
- **Prova** = una sequenza di applicazione di regole di inferenza

Può utilizzare regole di inferenza come operatori in un algoritmo standard di ricerca

- Tipicamente richiede la rappresentazione delle sentenze in una **forma normale**

Model checking

enumerazione della tabella di verità (sempre esponenziale in n)

backtracking migliorato, p.e., Davis–Putnam–Logemann–Loveland

ricerca euristica nello spazio dei modelli (corretta ma incompleta)

p.e., algoritmi hill-climbing basati sulla minimizzazione dei conflitti

Forward e backward chaining

Forma di Horn (ristretta)

KB = *congiunzione* di *Clausole di Horn*

Clausola di Horn =

◇ simbolo proposizionale; o

◇ (congiunzione di simboli) \Rightarrow simbolo

P.e., $C \wedge (B \Rightarrow A) \wedge (C \wedge D \Rightarrow B)$

Modus Ponens (per la Forma di Horn): completo per basi di conoscenza in forma di Horn

$$\frac{\alpha_1, \dots, \alpha_n, \quad \alpha_1 \wedge \dots \wedge \alpha_n \Rightarrow \beta}{\beta}$$

Può essere usato con **forward chaining** o **backward chaining**.

Questi algoritmi sono molto immediati e hanno complessità *lineare* in tempo

Forward chaining

Idea: applicare ogni regola le cui premesse sono soddisfatte in KB ,
aggiungere le conclusioni a KB , fino a quando non si trova la query

$$P \Rightarrow Q$$

$$L \wedge M \Rightarrow P$$

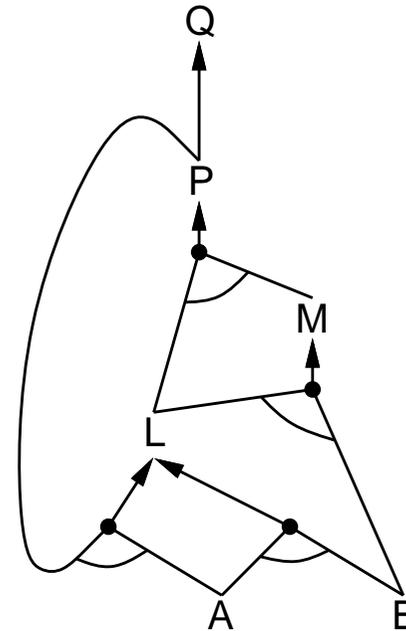
$$B \wedge L \Rightarrow M$$

$$A \wedge P \Rightarrow L$$

$$A \wedge B \Rightarrow L$$

A

B



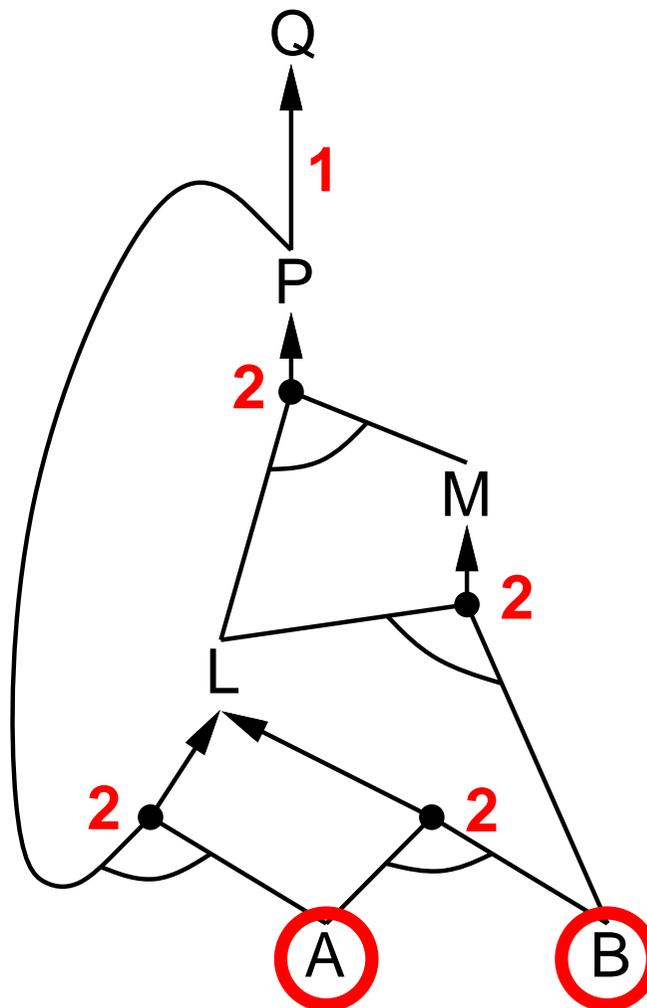
Algoritmo di Forward chaining

```
function CP-CA-IMPLICA?(KB, q) returns true or false
  local variables: conto, una tabella, indicizzata per clausola, che contiene
                  inizialmente il numero di premesse
                  inferiti, una tabella, indicizzata per simbolo, in cui ogni
                  elemento è inizialmente false
                  agenda, una lista di simboli, che contiene inizialmente
                  quelli noti come veri nella KB

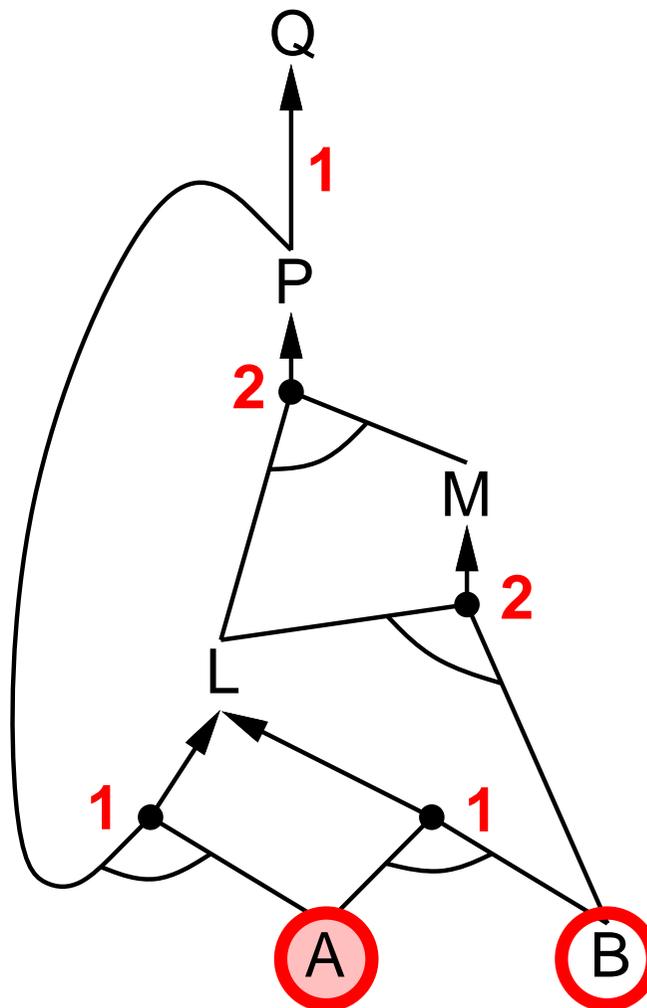
  while agenda non è vuota do
     $p \leftarrow \text{POP}(\textit{agenda})$ 
    unless inferiti[p] do
      inferiti[p]  $\leftarrow$  true
      for each clausola di Horn c in cui appare la premessa p do
        decrementa conto[c]
        if conto[c] = 0 then do
          if TESTA[c] = q then return true
          PUSH(TESTA[c], agenda)

  return false
```

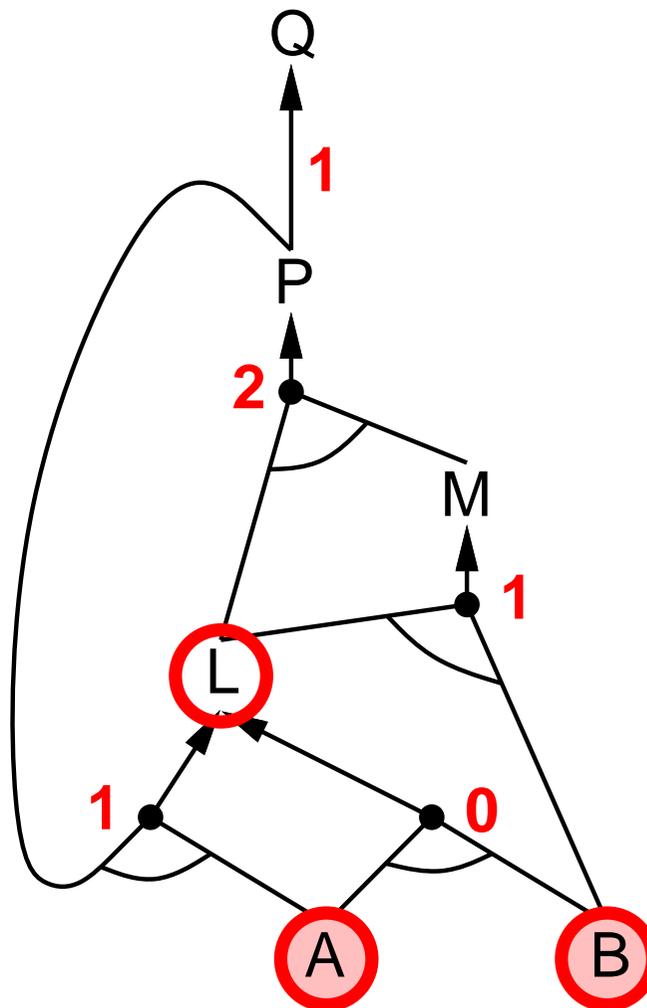
Esempio di Forward chaining



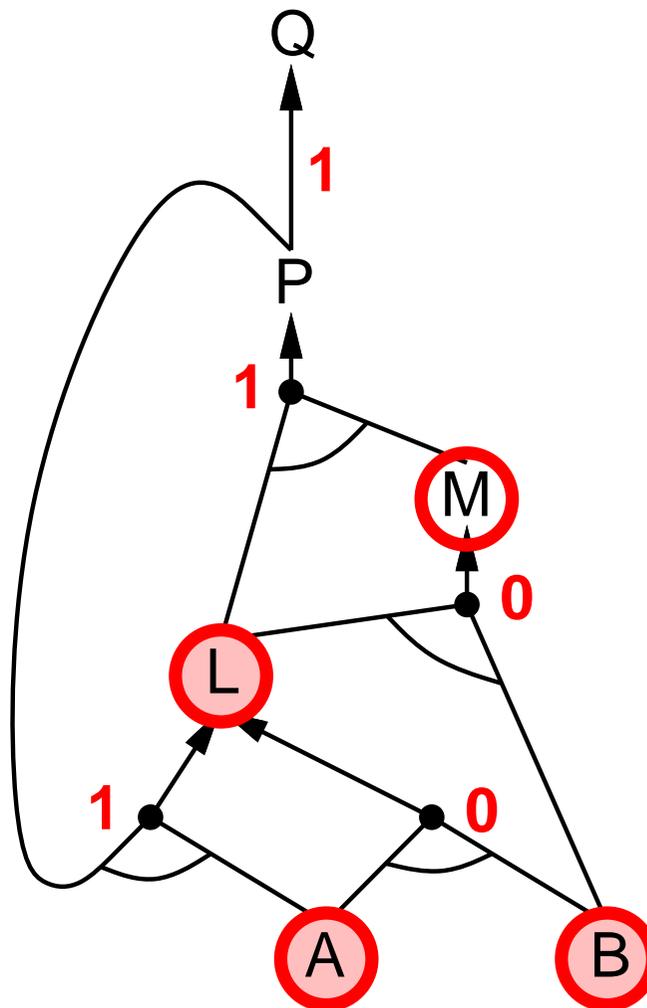
Esempio di Forward chaining



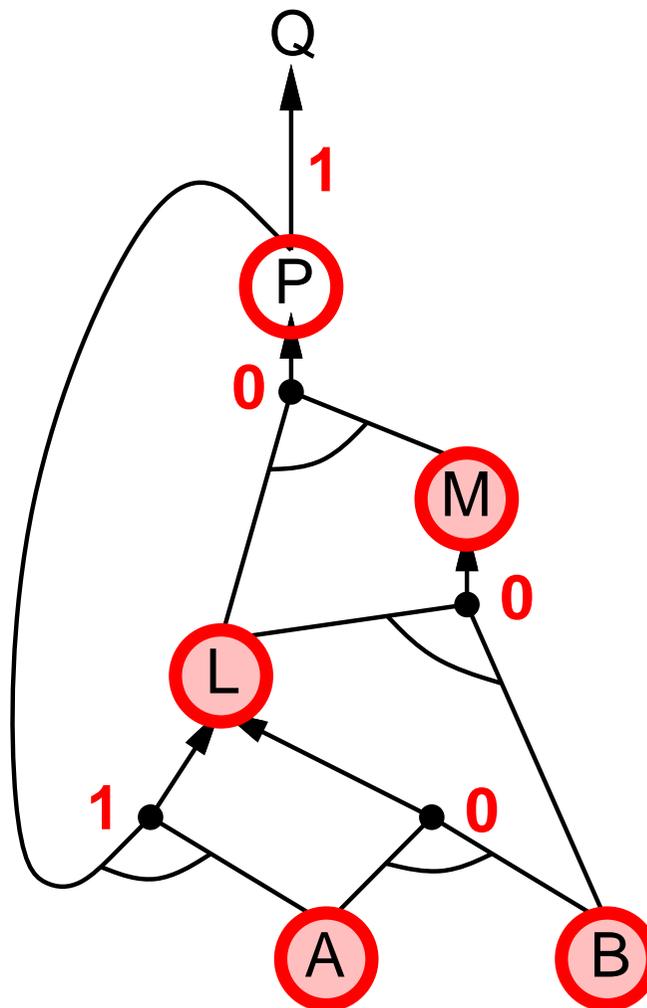
Esempio di Forward chaining



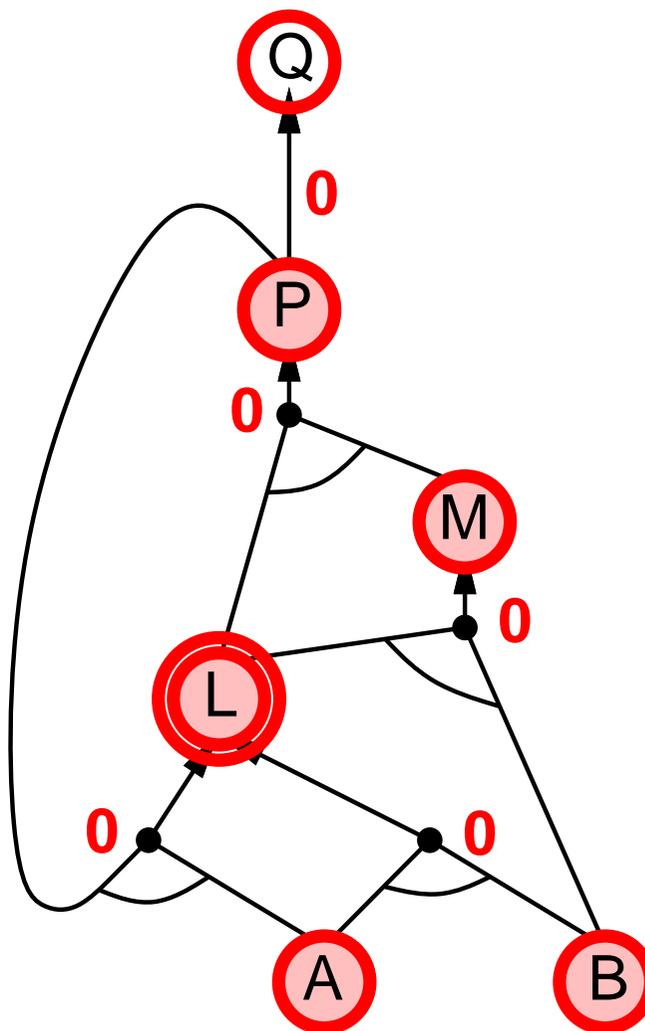
Esempio di Forward chaining



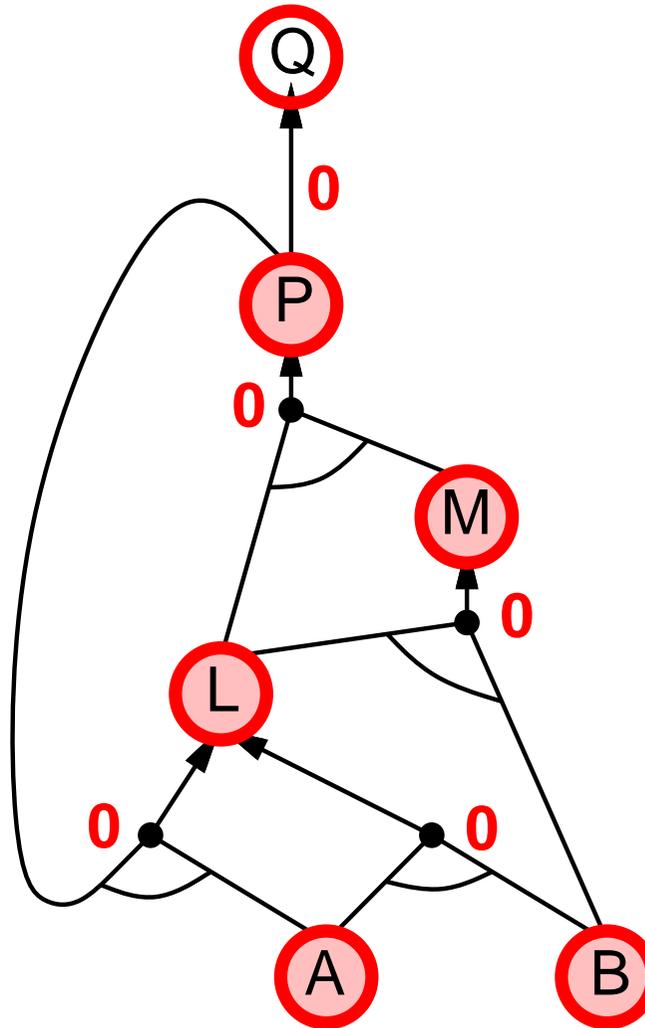
Esempio di Forward chaining



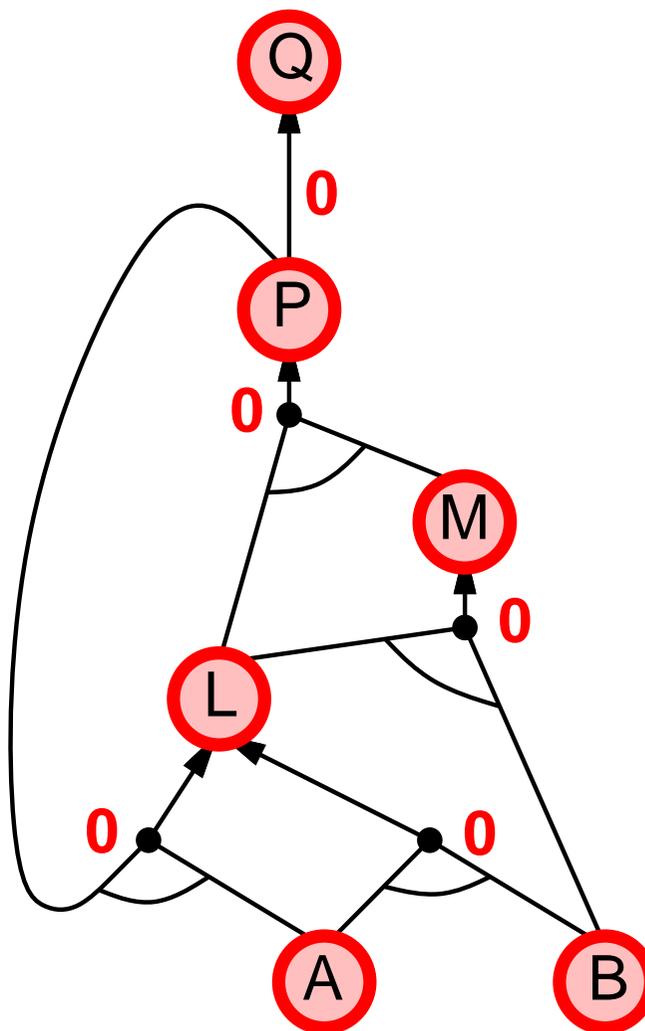
Esempio di Forward chaining



Esempio di Forward chaining



Esempio di Forward chaining



Prova di completezza

FC deriva ogni sentenza atomica che è conseguenza di KB

1. FC raggiunge un **punto fisso** dove nessuna nuova sentenza atomica è derivata

2. Si consideri lo stato finale come un modello m , assegnando vero/falso ai simboli

3. Ogni clausola nella KB originale è vera in m

Prova: Si supponga che una clausola $a_1 \wedge \dots \wedge a_k \Rightarrow b$ sia falsa in m

Allora $a_1 \wedge \dots \wedge a_k$ è vera in m e b è falsa in m

Perciò l'algoritmo non ha raggiunto un punto fisso!

4. Quindi m è un modello per KB

5. Se $KB \models q$, q è vera in **ogni** modello di KB , incluso m

Backward chaining

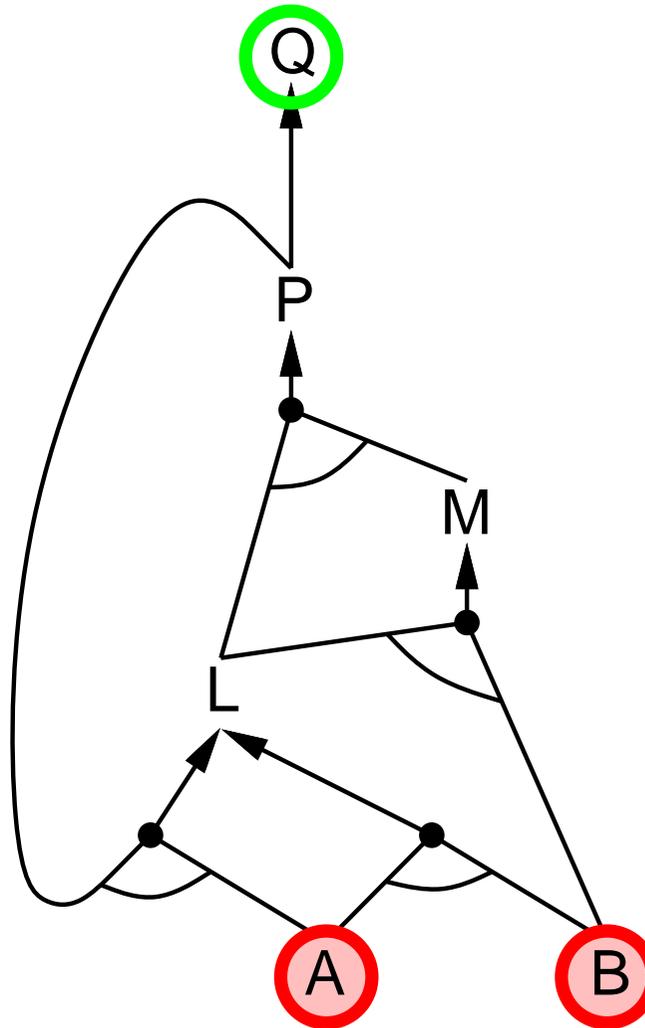
Idea: si lavora all'indietro (backwards) a partire dalla query q :
per provare q per mezzo di BC,
controlla se q è già conosciuta, o
prova tramite BC tutte le premesse di una regola che deriva q

Evitare cicli: controlla se un nuovo sottogoal è già presente nella pila dei goal

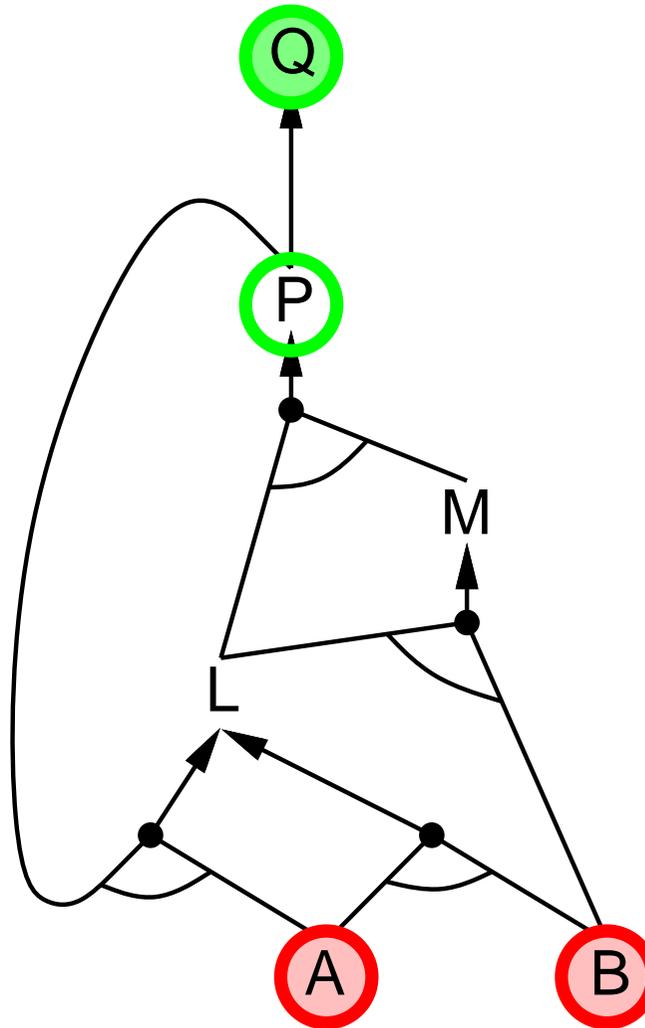
Evitare di ripetere del lavoro: controlla se un nuovo sottogoal

- 1) è stato già provato vero, o
- 2) è già fallito

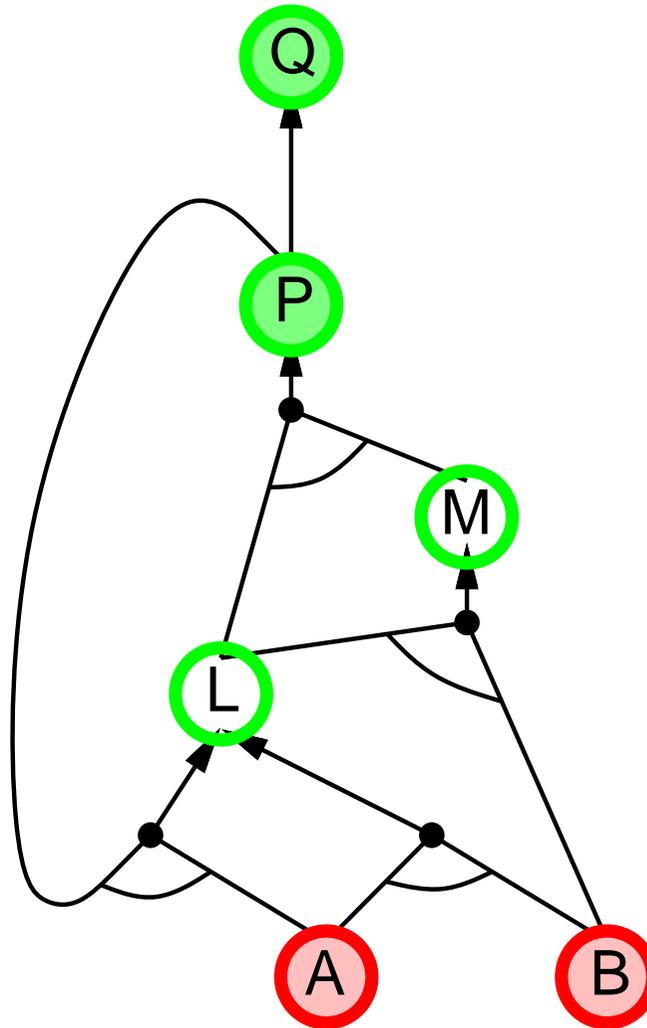
Esempio di Backward chaining



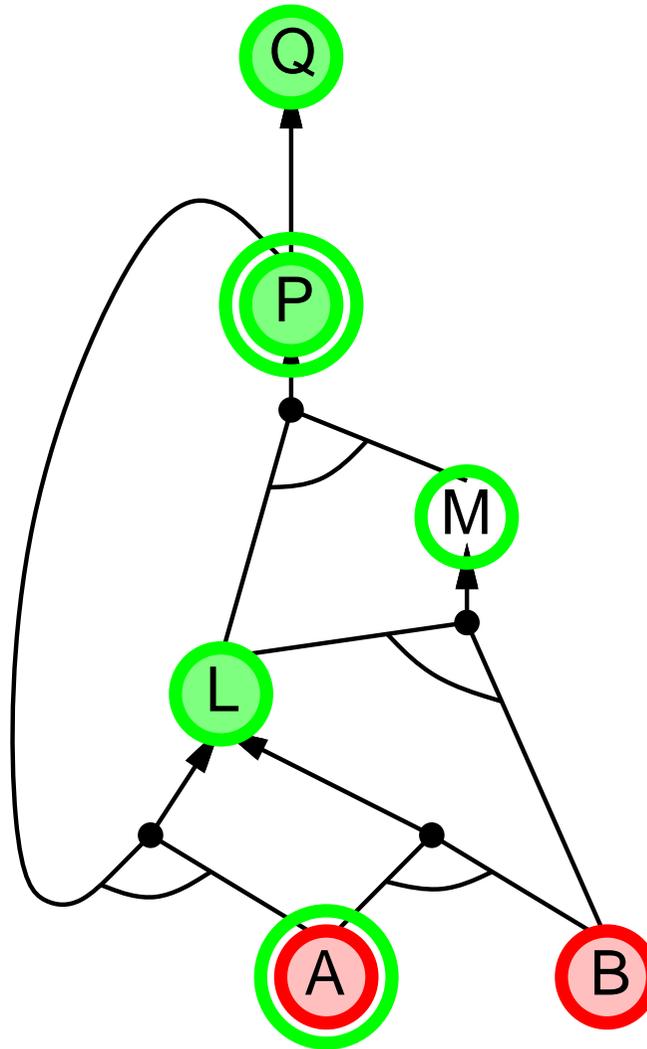
Esempio di Backward chaining



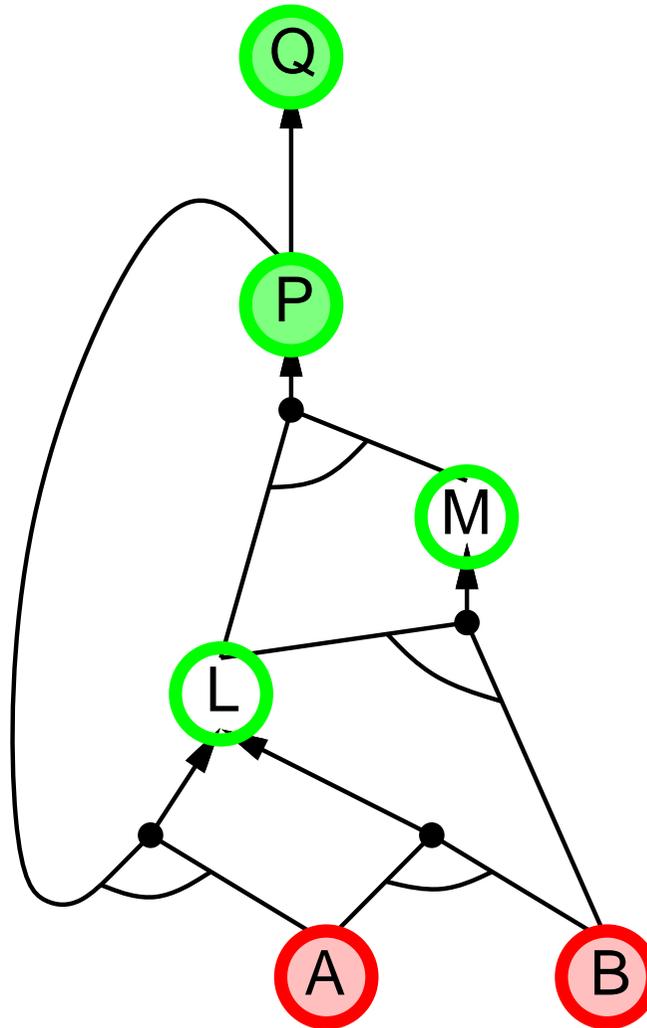
Esempio di Backward chaining



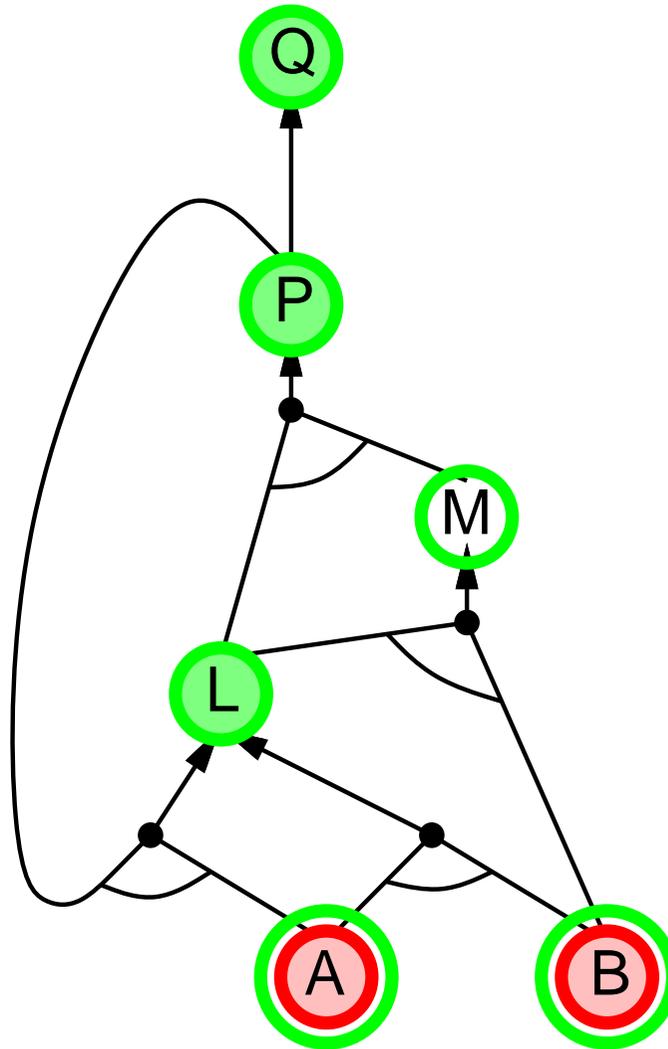
Esempio di Backward chaining



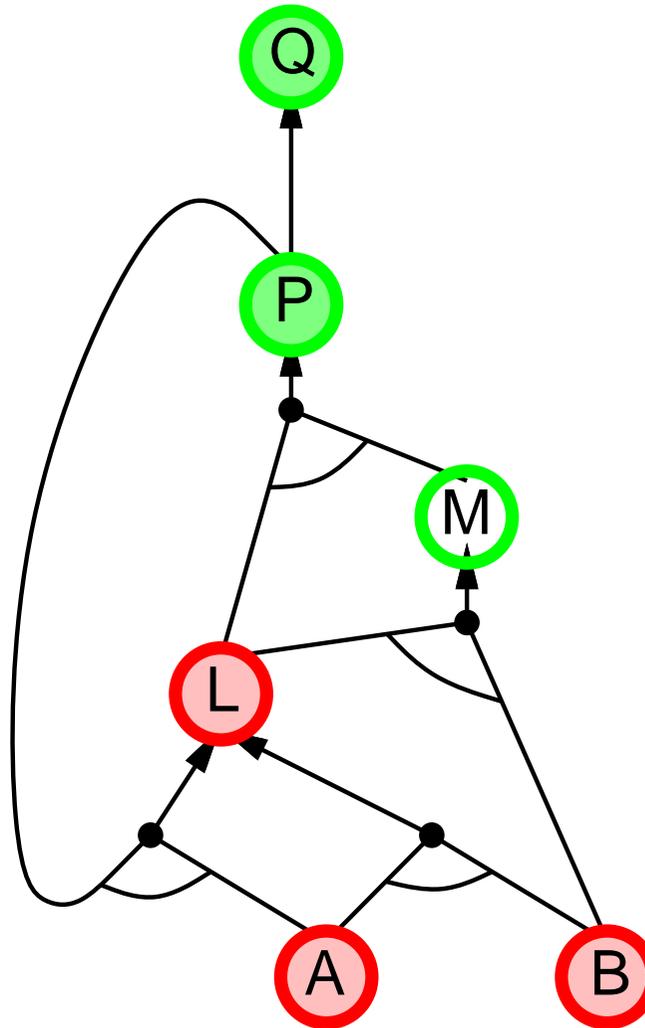
Esempio di Backward chaining



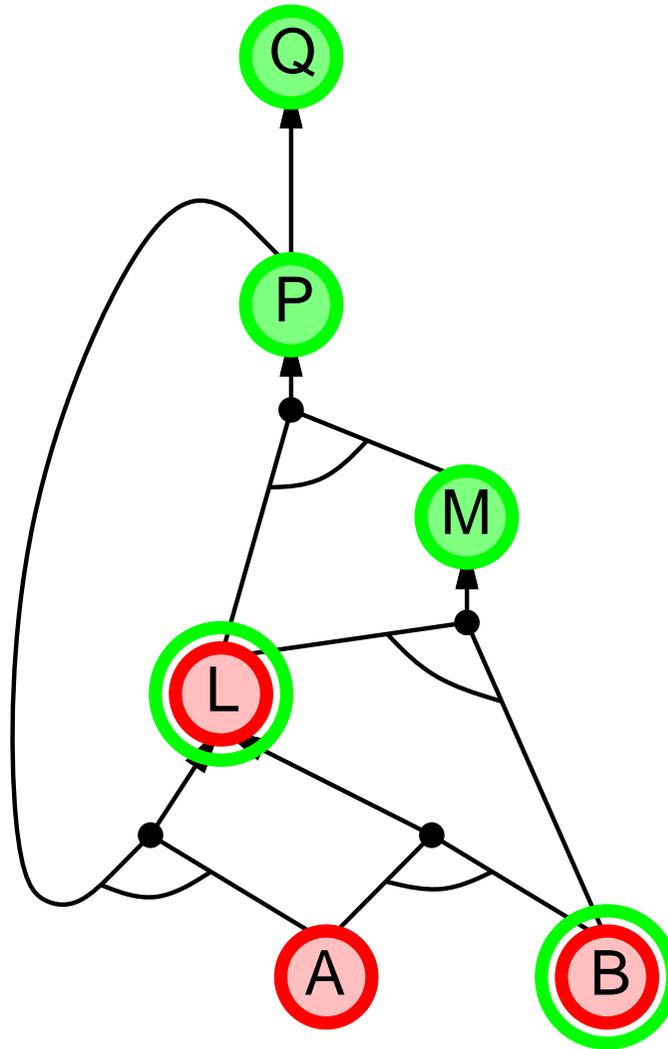
Esempio di Backward chaining



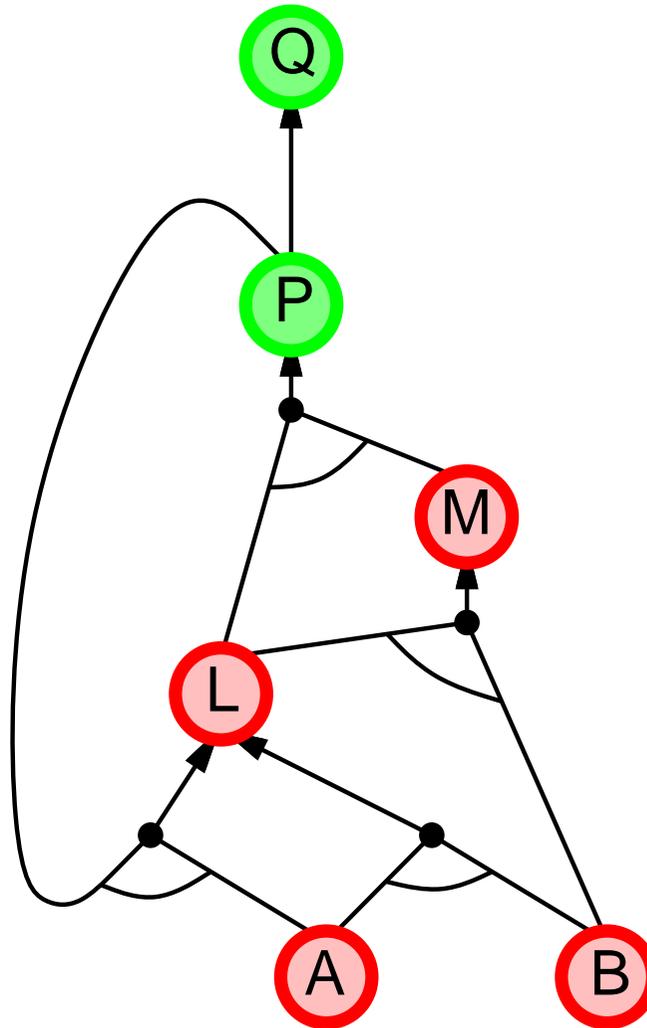
Esempio di Backward chaining



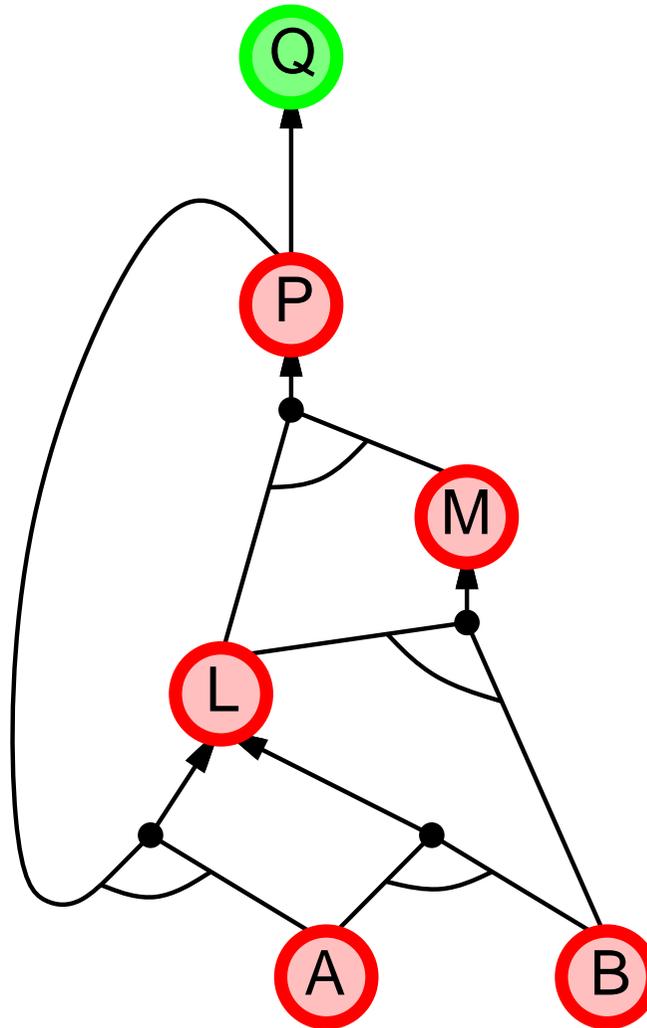
Esempio di Backward chaining



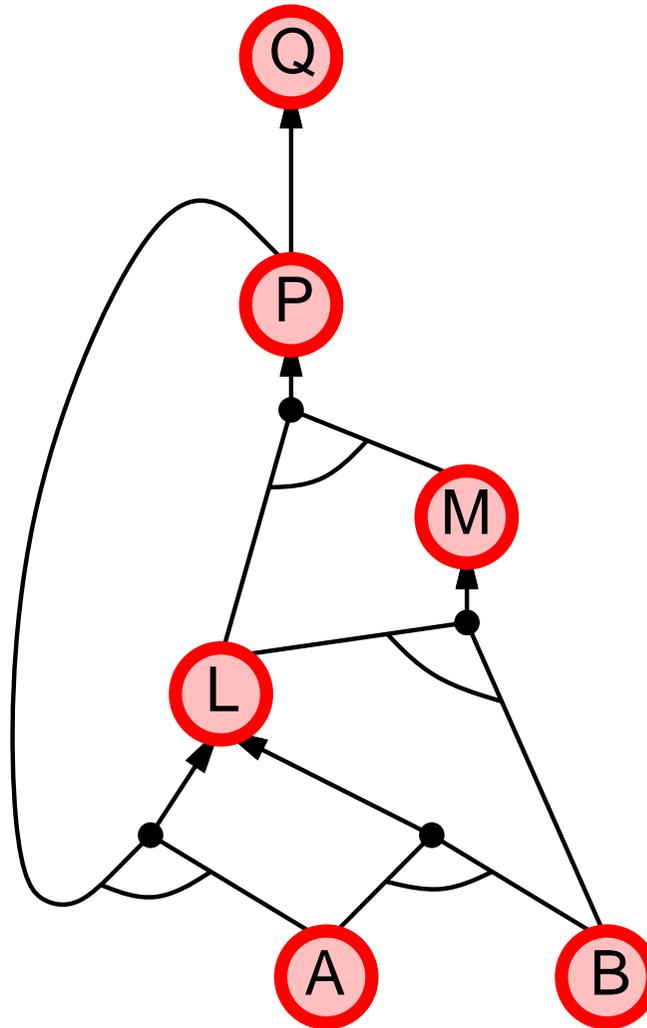
Esempio di Backward chaining



Esempio di Backward chaining



Esempio di Backward chaining



Forward contro backward chaining

FC è **data-driven** (guidato dai dati), conf. elaborazione inconscia, automatica,

p.e., riconoscimento di oggetti, decisioni di routine

Rischia di eseguire molto lavoro irrilevante per il goal

BC è **goal-driven** (guidato dal goal), appropriato per il problem-solving,
p.e., Dove sono le mie chiavi? Come faccio ad entrare nel dottorato di ricerca ?

La complessità di BC può essere **molto minore** che lineare nella dimensione di KB

Risoluzione

Forma Normale Congiuntiva (CNF—universale)

congiunzione di *disgiunzioni* di *letterali*
clausole

P.e., $(A \vee \neg B) \wedge (B \vee \neg C \vee \neg D)$

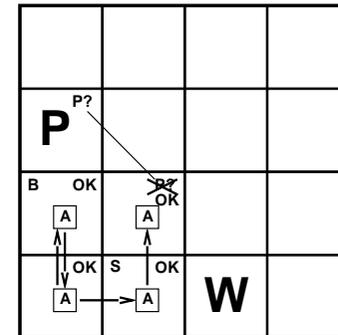
Risoluzione regola di inferenza (per CNF): completa per la logica proposizionale

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

dove l_i e m_j sono letterali complementari. P.e.,

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

La Risoluzione è corretta e completa
per la logica proposizionale



Conversione in CNF

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

1. Eliminare \Leftrightarrow , rimpiazzando $\alpha \Leftrightarrow \beta$ con $(\alpha \Rightarrow \beta) \wedge (\beta \Rightarrow \alpha)$.

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

2. Eliminare \Rightarrow , rimpiazzando $\alpha \Rightarrow \beta$ con $\neg\alpha \vee \beta$.

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

3. Spostare \neg all'interno usando le regole di de Morgan e la doppia negazione:

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

4. Applicare la legge distributiva (\vee su \wedge) e portare tutto su un livello:

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

Algoritmo di Risoluzione

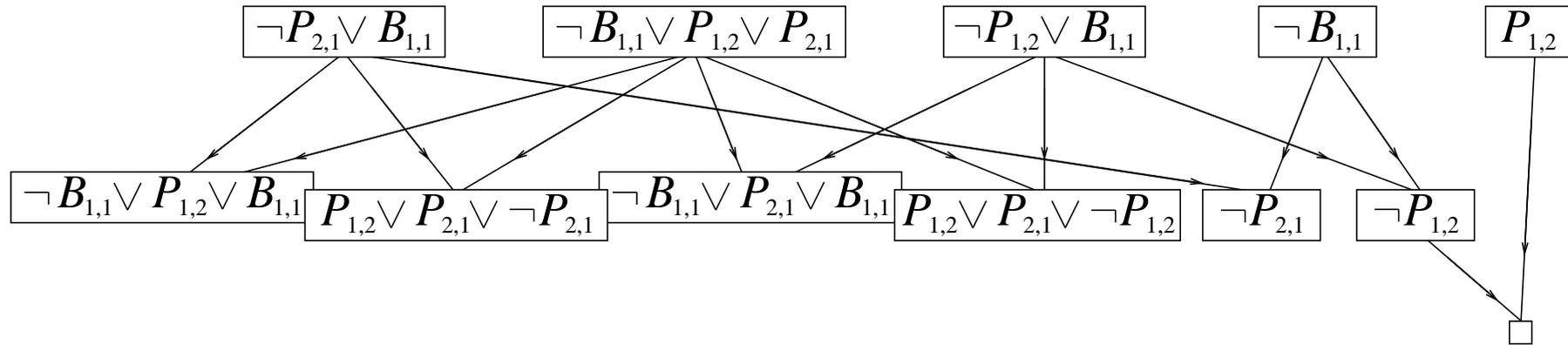
Prova per contraddizione, cioè, mostra che $KB \wedge \neg\alpha$ è insoddisfacibile

```
function CP-RISOLUZIONE( $KB, \alpha$ ) returns true oppure false  
clause  $\leftarrow$  l'insieme di clausole nella rappresentazione CNF di  $KB \wedge \neg\alpha$   
new  $\leftarrow$  { }  
loop do  
  for each  $C_i, C_j$  in clause do  
    resolvents  $\leftarrow$  CP-RISOLVI( $C_i, C_j$ )  
    if resolvents contiene la clausola vuota then return true  
    new  $\leftarrow$  new  $\cup$  resolvents  
  if new  $\subseteq$  clause then return false  
  clause  $\leftarrow$  clause  $\cup$  new
```

CP-RISOLVI(C_i, C_j) restituisce l'insieme dei risolventi ottenuti applicando la regola di risoluzione in tutti i modi possibili per le clausole C_i e C_j

Esempio di Risoluzione

$$KB = (B_{1,1} \Leftrightarrow (P_{1,2} \vee P_{2,1})) \wedge \neg B_{1,1} \quad \alpha = \neg P_{1,2}$$



Completezza della Risoluzione

Il teorema di completezza per la risoluzione nella logica proposizionale è chiamato **ground resolution theorem**:

Se un insieme di clausole S è insoddisfacibile, allora la chiusura della risoluzione di tali clausole $RC(S)$ contiene la clausola vuota.

La chiusura della risoluzione di un insieme di clausole corrisponde all'insieme ottenuto come punto fisso dalla applicazione ripetuta della risoluzione. In caso di terminazione con fallimento dell'algoritmo visto, corrisponde all'insieme *clauses* alla fine della esecuzione.

La dimostrazione del teorema si ottiene dimostrando quanto segue:

Se la chiusura $RC(S)$ non contiene la clausola vuota, allora S è soddisfacibile.

vediamo la dimostrazione di questa affermazione ...

Completezza della Risoluzione

Se $RC(S)$ non contiene la clausola vuota, allora si può costruire un modello per S . Infatti, si dia un ordine arbitrario ai simboli di predicato che compaiono in S , ottenendo P_1, P_2, \dots, P_k . Poi si segua la seguente procedura per costruire il modello:

for $i = 1$ to k

- se esiste una clausola in $RC(S)$ contenente $\neg P_i$ tale che tutti gli altri letterali della clausola sono falsi a causa del valore di verità già assegnato ai P_1, P_2, \dots, P_{i-1} , allora assegna valore di verità falso a P_i
- altrimenti assegna valore di verità vero a P_i

Mostriamo ora che tale procedura termina sempre con un modello per S .

Facciamo tale dimostrazione per induzione su i : supponiamo che sia possibile costruire il modello parziale per i simboli fino a P_{i-1} e mostriamo che tale modello può essere esteso per i simboli fino a P_i

Completezza della Risoluzione

Caso base $i = 1$.

In questo caso, in S non possono necessariamente essere presenti simultaneamente sia la clausola P_1 e $\neg P_1$, perchè altrimenti $RC(S)$ conterrebbe la clausola vuota. Quindi per $i = 1$ la procedura descritta si può applicare senza problemi: $P_1 \leftarrow \textit{false}$ se è presente $\neg P_1$, altrimenti $P_1 \leftarrow \textit{vero}$.

Ipotesi induttiva vera per $i - 1$.

Consideriamo una clausola C in $RC(S)$ che contiene P_i . Si hanno problemi ad assegnare un valore di verità a P_i solo se

- $C \equiv B \vee \neg P_i$, con B clausola che contiene solo simboli P_j con $j < i$
- esiste in $RC(S)$ una clausola $C' \equiv B' \vee P_i$ con B' clausola che contiene solo simboli P_j con $j < i$

Completezza della Risoluzione

Ma se questo succede, in $RC(S)$ deve essere presente anche la clausola $B \vee B'$ (che contiene solo simboli P_j con $j < i$) altrimenti $RC(S)$ non è la chiusura, ed a causa della ipotesi induttiva, l'assegnamento parziale fino a P_{i-1} non può rendere falsa sia B che B' . Quindi, se è falsa B , $P_i \leftarrow falso$, se invece è falsa B' , $P_i \leftarrow vero$, ottenendo un modello parziale per i simboli fino all'indice i .

Si conclude che quando i raggiunge k , otteniamo un modello completo per S e quindi abbiamo dimostrato che S è soddisfacibile.

Riassunto

Gli agenti logici applicano *l'inferenza* ad una *base di conoscenza* per derivare nuova informazione e prendere decisioni

Il mondo dei Wumpus world richiede la capacità di rappresentare informazione parziale e negata, ragionamento per casi, etc.

Forward e Backward chaining sono lineari, e completi per clausole di Horn
La Risoluzione è completa per la logica proposizionale

La logica proposizionale manca di potere espressivo