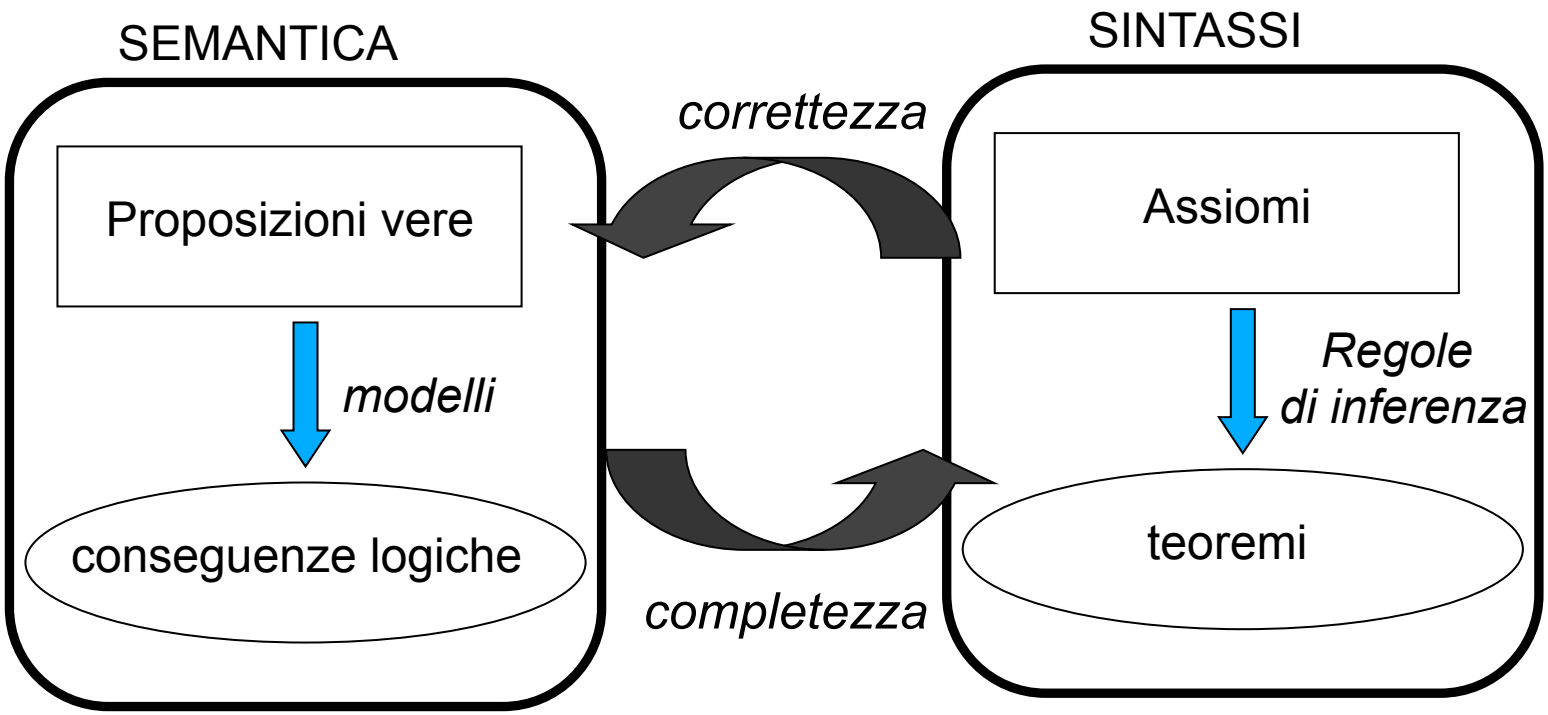
Abbiamo visto…

- Gli agenti logici applicano inferenze a una base di conoscenza per derivare nuove informazioni.

- Concetti base della logica:
  - sintassi: struttura formale delle sentenze
  - semantica: verità di sentenze rispetto ad interpretazioni/modelli
  - conseguenza logica (entailment): sentenza necessariamente vera data un'altra sentenza
  - inferenza: derivare (sintatticamente) sentenze da altre sentenze
  - correttezza (soundness): la derivazione produce solo sentenze che sono conseguenza logica.
  - Completezza (completeness): la derivazione può prdurre tutte le conseguenze logiche.

CORRETTEZZA E COMPLETEZZA (2)



INSIEMI DI FORMULE (*recap*)

- Un **insieme** di formule chiuse del primo ordine S è **soddisfacibile** se esiste una interpretazione I che soddisfa **tutte le formule** di S (cioè che è un modello per ciascuna formula di S). Tale interpretazione è detta modello di S.
- Esempi di insiemi di formule insoddisfacibili sono:
  - S1={~ (∃X ∀Y p(X,Y)), ∃X ∀Y p(X,Y)}
  - S2={p(s(0),0)⇒p(0,s(0)), p(s(0),0), ~p(0,s(0))}
- Un **insieme** di formule chiuse del primo ordine S è **insoddisfacibile** se non esiste alcun modello per **tutte le formule** di S.

CONSEGUENZA LOGICA (2)

- Una formula F **segue logicamente** (o è conseguenza logica) da un insieme di formule S (e si scrive **S I= F**), se e solo se ogni interpretazione I che è un modello per S, è anche un modello per F.

Proprietà (*ci torneranno utili più avanti*):

- Se una fbf F segue logicamente da S (S I= F), allora l'insieme S ∪ {~F} è insoddisfacibile.

CONSEGUENZA LOGICA (2)

- Una formula F **segue logicamente** (o è conseguenza logica) da un insieme di formule S (e si scrive **S I= F**), se e solo se ogni interpretazione I che è un modello per S, è anche un modello per F.

Proprietà (*ci torneranno utili più avanti*):

- Se una fbf F segue logicamente da S (S I= F), allora l'insieme S ∪ {~F} è insoddisfacibile.
- **Viceversa, se S ∪ {~F} è insoddisfacibile (e S era soddisfacibile), allora F segue logicamente da S**.
- Poiché è difficile lavorare a livello semantico (interpretazioni, modelli sono spesso infiniti e/o in numero infinito), allora …
- **… si lavora a livello sintattico**, ma **operando per refutazione**

SISTEMI DI REFUTAZIONE

- I sistemi di refutazione si basano su questa proprietà: per dimostrare S |= F, supposto S soddisfacibile, è sufficiente dimostrare che S∪{~F} è insoddisfacibile.
- Problema (già visto con Calcolo dei Predicati):

  Determinare se una formula F segue logicamente da S (ovvero che S ∪ {~F} è insoddisfacibile) **utilizzando solo semplici trasformazioni sintattiche (regole di inferenza)**, possibilmente ripetitive e quindi automatizzabili, e non introducendo concetti quali significato o interpretazione o modello

→ ***Principio di risoluzione***

(regola di inferenza – corretta e completa – per logica a clausole)

Proprietà

- Se la *proof theory* è corretta e completa è garantita l'equivalenza tra l'aspetto sintattico e semantico

$$\mathbf{T \models F \quad \Leftrightarrow \quad T \vdash F}$$

- Abbiamo detto che se F è conseguenza logica di T, $T \cup \{\sim F\}$ è insoddisfacibile :

$$\mathbf{T \models F \quad \Leftrightarrow \quad T \cup \{\sim F\}}$$

Sommario

T |= F ⇔ T|- F

T |= F ⇔ T ∪{~F} insoddisfacibile

- Applichiamo una regola di inferenza (**Principio di risoluzione**) che lavora per per refutazione, in teorie con formule a clausole, per dimostrare che da T ∪{~F} segue la contraddizione logica
- Correttezza e completezza
- Vediamo prima il caso proposizionale, poi quello del primo ordine

Logica Proposizionale:

- E' la logica più semplice, ma non molto espressiva. Non possiamo esprimere variabili (solo enumerazione di tutti gli elementi)

- Se S, S_1, S_2 sono sentenze allora sono anche sentenze:

  - $\neg S$ (negazione)
  - $S_1 \wedge S_2$ (congiunzione)
  - $S_1 \vee S_2$ (disgiunzione)
  - $S_1 \Rightarrow S_2$ (implicazione)
  - $S_1 \Leftrightarrow S_2$ (bicondizionale)

Dimostrazioni in logica proposizionale

- Vedremo la dimostrazione basata su:
  - Risoluzione (corretta e completa per clausole generali)
  - Forward chaining (corretta e completa per clausole Horn)
  - Backward chaining (corretta e completa per clausole Horn)

- Una qualunque fbf della logica proposizionale si può trasformare in un equivalente insieme di clausole generali (formule SP o PS, vedi reti logiche ed algebra di Boole).

IL PRINCIPIO DI RISOLUZIONE

- Sistema di deduzione per la logica a clausole per il quale valgono interessanti proprietà.

- Regola di inferenza**: Principio di Risoluzione** [Robinson 65], che si applica a teorie del primo ordine in **forma a clausole**.

- È la regola di inferenza base utilizzata nella programmazione logica.

CLAUSOLE

- Una **clausola** è una disgiunzione di letterali (cioè formule atomiche negate e non negate), in cui tutte le variabili sono quantificate universalmente in modo implicito.
- Una clausola generica può essere rappresentata come la disgiunzione:

  $A_1 \vee A_2 \vee \ldots \vee A_n \vee \sim B_1 \vee \ldots \vee \sim B_m$

  dove A_i (i=1,..,n) e B_j (j=1,..,m) sono atomi.
- Una clausola nella quale non compare alcun letterale, sia positivo sia negativo, è detta clausola vuota e verrà indicata con □. interpretato come contraddizione: disgiunzione falso v ~vero
- Un sottoinsieme delle clausole è costituito dalle **clausole definite**, nelle quali si ha sempre un solo letterale positivo:

  $A_1 \vee \sim B_1 \vee \ldots \vee \sim B_m$

CLAUSOLE (2)

- La Conjunctive Normal Form (CNF) è di fatto **la forma normale di riferimento (AND di OR)**
- Una formula in CNF è una congiunzione di disgiunzioni di letterali (atomi o loro negazione)
- **Ogni formula della logica proposizionale è logicamente equivalente a una congiunzione di disgiunzioni di letterali**
- I passi consistono nell'eliminare: $\Leftrightarrow$, poi $\Rightarrow$, applicare $\neg$ a atomi (De Morgan e/o idempotenza del $\neg$, distribuzione dell'$\vee$ sull'$\wedge$)
- La formula CNF è quindi la congiunzione di clausole (che sono formule di sole disgiunzioni)

IL PRINCIPIO DI RISOLUZIONE

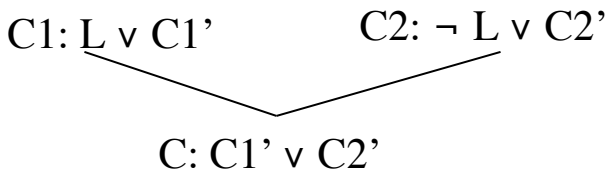
- Il principio di risoluzione, che si applica a formule in forma a clausole, è utilizzato dalla maggior parte dei risolutori automatici di teoremi.
- **Logica Proposizionale:** clausole prive di variabili.
- Siano C_1 e C_2 due clausole in logica poposizionale:

  $C_1 = A_1 v ... v A_n$ $\qquad\qquad$ $C_2 = B_1 v ... v B_m$

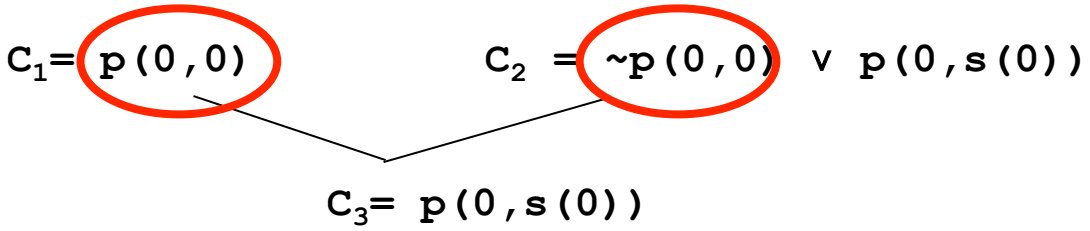
- Se esistono in C_1 e C_2 due letterali **opposti**, A_i e B_j, ossia tali che $A_i = \sim B_j$, allora da C_1 e C_2, (clausole **parent**) si può derivare una nuova clausola C_3, denominata **risolvente**, della forma:

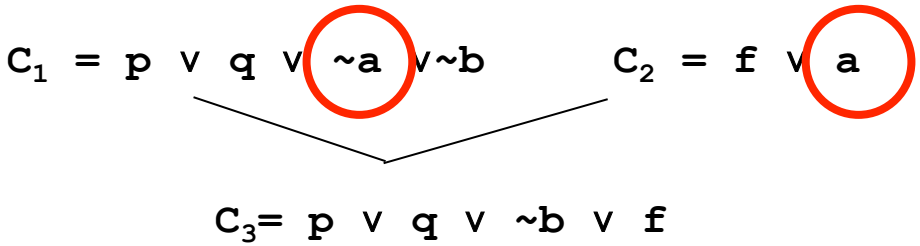
  $C_3 = A_1 v .. v A_{i-1} v A_{i+1} v .. v A_n v B_1 v .. v B_{j-1} v B_{j+1} v ... v B_m$

- **C_3 è conseguenza logica di $C_1 \cup C_2$.**



ESEMPI





- Si usa inoltre **fattorizzazione** (che da: A v A v **Rest** inferisce A v **Rest**) rimuovendo dalle clausole eventuali copie di letterali

Risoluzione *vs* Modus Ponens

- Risoluzione

$$\frac{A_1 \vee \ldots \vee A_n \qquad B_1 \vee \ldots \vee B_m \qquad A_i = \sim B_j}{A_1 \vee .. \vee A_{i-1} \vee A_{i+1} \vee .. \vee A_n \vee B_1 \vee .. \vee B_{j-1} \vee B_{j+1} \vee \ldots \vee B_m}$$

- Modus Ponens

$$\frac{A \qquad \sim A \vee B}{B}$$

DIMOSTRAZIONE PER CONTRADDIZIONE ATTRAVERSO LA RISOLUZIONE (1)

- Dati gli assiomi propri H di una teoria e una formula F, derivando da $H \cup \{\sim F\}$ la contraddizione logica si dimostra che F è un teorema della teoria.
- 1) Ridurre H e il teorema negato ~F in forma a clausole.

  H trasformato nell'insieme di clausole H^C: $H \rightarrow H^C$

  F negata e trasformata nell'insieme di clausole F^C: $\sim F \rightarrow F^C$
- 2) **All'insieme $H^C \cup F^C$ si applica la risoluzione**

  Se F è un teorema della teoria, allora la risoluzione deriva la contraddizione logica (**clausola vuota**) in un numero finito di passi.
- **Contraddizione:** Nella derivazione compariranno due clausole del tipo A e ~A

DIMOSTRAZIONE PER CONTRADDIZIONE ATTRAVERSO LA RISOLUZIONE (2)

- Per dimostrare F, il metodo originario (Robinson, 1965) procede generando i risolventi per **tutte le coppie** di clausole dell'insieme di partenza $C_0 = H^C \cup F^C$ che sono aggiunti a C_0. Procedimento iterato, fino a derivare, se è possibile, la clausola vuota.
- 1. $C_{i+1} = C_i \cup \{\text{risolventi delle clausole di } C_i\}$
- 2. Se C_{i+1} contiene la clausola vuota, termina.
- Altrimenti ripeti il passo 1.
- E' una regola di inferenza **corretta e completa** (deriva la clausola vuota se e solo se $H \cup \{\sim F\}$ è insoddisfacibile).

ESEMPIO

$H = \{(a \rightarrow c \vee d) \wedge (a \vee d \vee e) \wedge (a \rightarrow \sim c)\}$
$F = \{d \vee e\}$

- Data la teoria H (insieme di assiomi, assunti veri), F è un teorema?
- Lo dimostro per contraddizione, negando F, aggiungendolo ad H, trasformando tutte le formule in clausole e cercando di derivare la clausola vuota

ESEMPIO

$H = \{(a \rightarrow c \vee d) \wedge (a \vee d \vee e) \wedge (a \rightarrow \sim c)\}$

$F = \{d \vee e\}$ $\sim F = \{\sim d \wedge \sim e\}$ (cioe' $\sim(d \vee e)$)

- La trasformazione in clausole di H e ~F produce:

$H^C = \{\sim a \vee c \vee d,\ a \vee d \vee e,\ \sim a \vee \sim c\}$

$F^C = \{\sim d,\ \sim e\}$

Si vuole dimostrare che $H^C \cup F^C$:

$\{\sim a \vee c \vee d$ (1)

$a \vee d \vee e,$ (2)

$\sim a \vee \sim c,$ (3)

$\sim d,$ (4)

$\sim e\}$ (5)

- è contraddittorio.

ESEMPIO

$H^C \cup F^C$:

{~a v c v d, (1)

a v d v e, (2)

~a v ~c, (3)

~d, (4)

~e} (5)

- **Tutti i possibili risolventi al passo 1 sono:**

{c v d v e, (6) da (1) e (2)

d v e v ~c, (7) da (2) e (3)

~a v c, (8) da (1) e (4)

a v e, (9) da (2) e (4)

a v d, (10) da (2) e (5)

~a v d} (11) da (1) e (3)

- **Al passo 2, da (10) e (11) viene derivato il risolvente:**

d (12)

e al passo 3, da (4) e (12) viene derivata anche la clausola vuota.

Algoritmo di Risoluzione per PL (Logica Proposizionale)

- *KB* base di conoscenza, α query
- Per contraddizione, i.e., *KB*∧¬α è insoddisfacibile

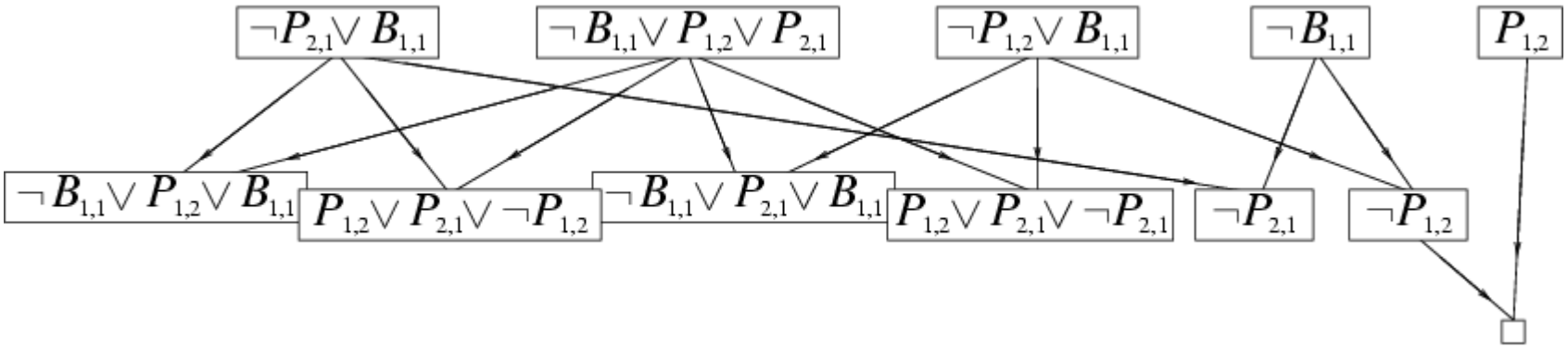
function PL-RESOLUTION(KB, α) **returns** *true* or *false*

clauses ← the set of clauses in the CNF representation of $KB \wedge \neg\alpha$
new ← { }
loop do
  for each C_i, C_j **in** *clauses* **do**
    resolvents ← PL-RESOLVE(C_i, C_j)
    if *resolvents* contains the empty clause **then return** *true*
    new ← *new* ∪ *resolvents*
  if *new* ⊆ *clauses* **then return** *false*
  clauses ← *clauses* ∪ *new*

Esempio di Risoluzione

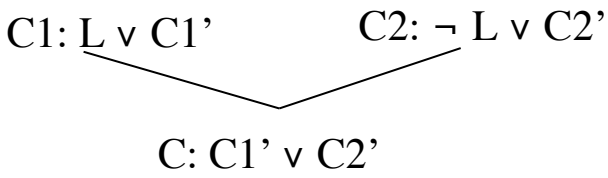
- *KB* = ($B_{1,1} \Leftrightarrow (P_{1,2} \vee P_{2,1})) \wedge \neg B_{1,1}$
- $\alpha = \neg P_{1,2}$

- *Convertendo KB*$\wedge\neg\alpha$ in CNF si ottengono le clausole riportate nella parte superiore della figura:



Teorema di risoluzione ground

- La risoluzione è una regola di inferenza corretta:



- In ogni modello di C1 e C2, o C1' è vera o è vera C2'
- L'applicazione ripetuta dell'algoritmo di Risoluzione per PL produce la **chiusura della teoria**. Vale inoltre:
- Teorema (completezza)

  Se un insieme di clausole è insoddisfacilbile, allora la sua chiusura per risoluzione contiene la clausola vuota

$H \cup \{\sim F\}$ è insoddisfacibile se e solo se $H^C \cup F^C$ è insoddisfacibile

Calcolo dei predicati del I ordine

- Calcolo per Logica dei predicati del I ordine (First Order Logic)
- Variabili quantificate (universalmente o esistenzialmente) nelle formule
- Due regole di inferenza

- **Modus Ponens (MP):**

$$\frac{A,\ A \rightarrow B}{B}$$

- **Specializzazione (Spec)**:

$$\frac{\forall X \quad A}{A(t)}$$

- Estensione del Principio di risoluzione per trattare variabili quantificate

Altre inferenze per PL (Propositional Logic): Forward e backward chaining

- Horn Form (sottoinsieme della logica proposizionale)
  KB = congiunzione di clausole di Horn
  - Clausole di Horn =
    - Proposizioni atomiche; o
    - (congiunzione di proposizioni atomiche) ⇒ proposizione atomica
  - Al massimo un letterale positivo
  - E.g. KB = C ∧ (B ⇒ A) ∧ (C ∧ D ⇒ B)

- Modus Ponens (per Horn): completo per Horn KB:

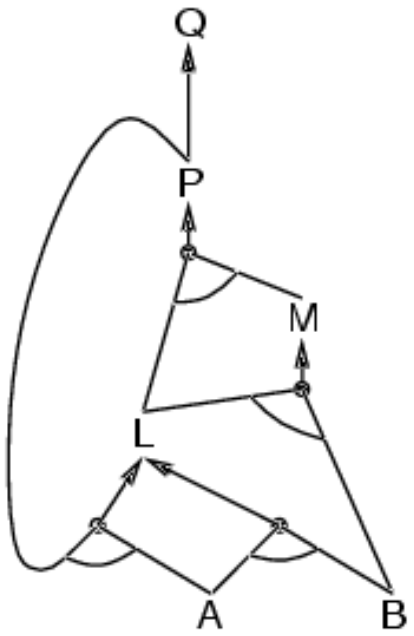
$$\frac{\alpha_1, \ldots, \alpha_n, \qquad \alpha_1 \wedge \ldots \wedge \alpha_n \Rightarrow \beta}{\beta}$$

- Può essere usato sia per forward chaining o backward chaining.
  Algoritmi molto naturali e con **complessità lineare in tempo**

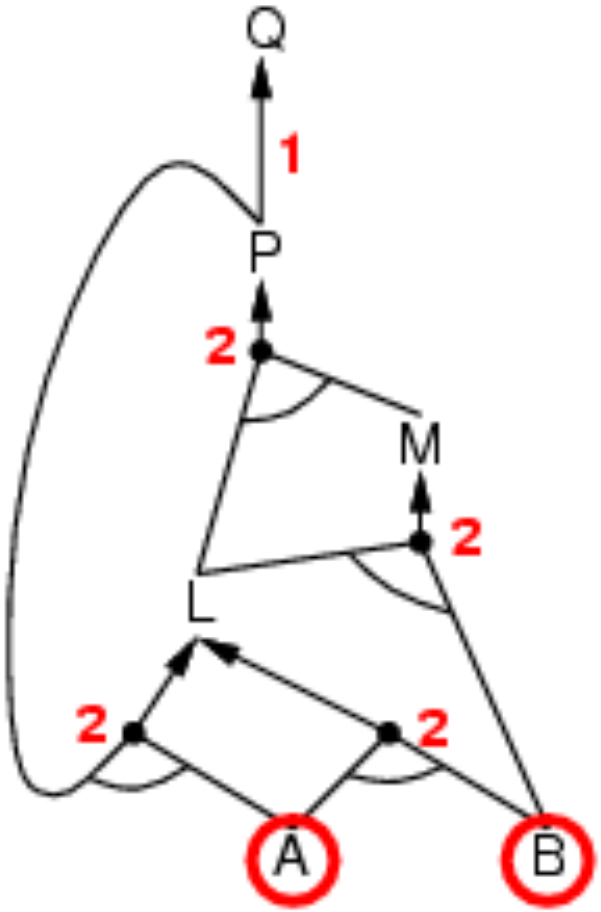
Forward chaining

- Idea: applica tutte le regole le cui premesse sono soddisfatte nella *KB*,
  - Aggiungi le conclusioni alla *KB*, fino a trovare la query

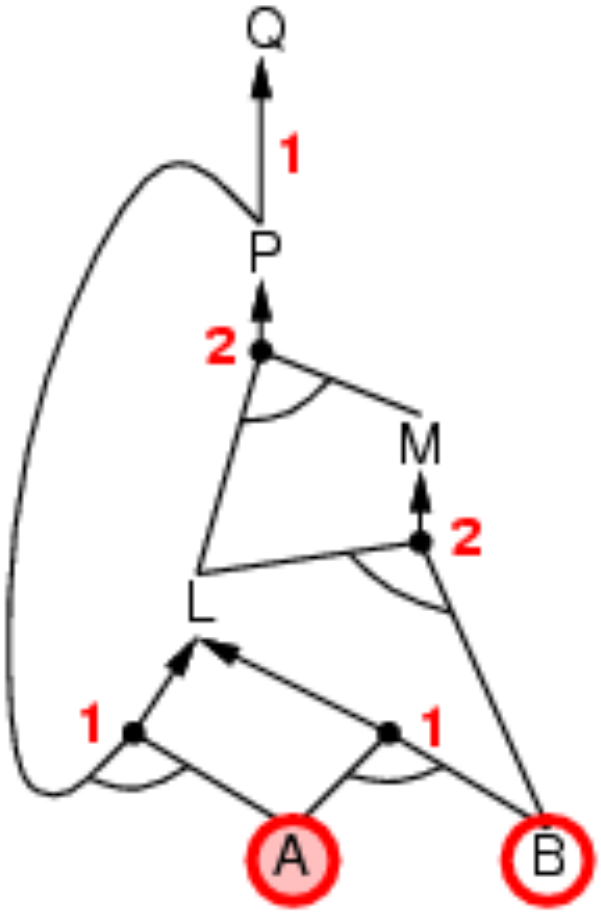
$$P \Rightarrow Q$$
$$L \wedge M \Rightarrow P$$
$$B \wedge L \Rightarrow M$$
$$A \wedge P \Rightarrow L$$
$$A \wedge B \Rightarrow L$$
$$A$$
$$B$$



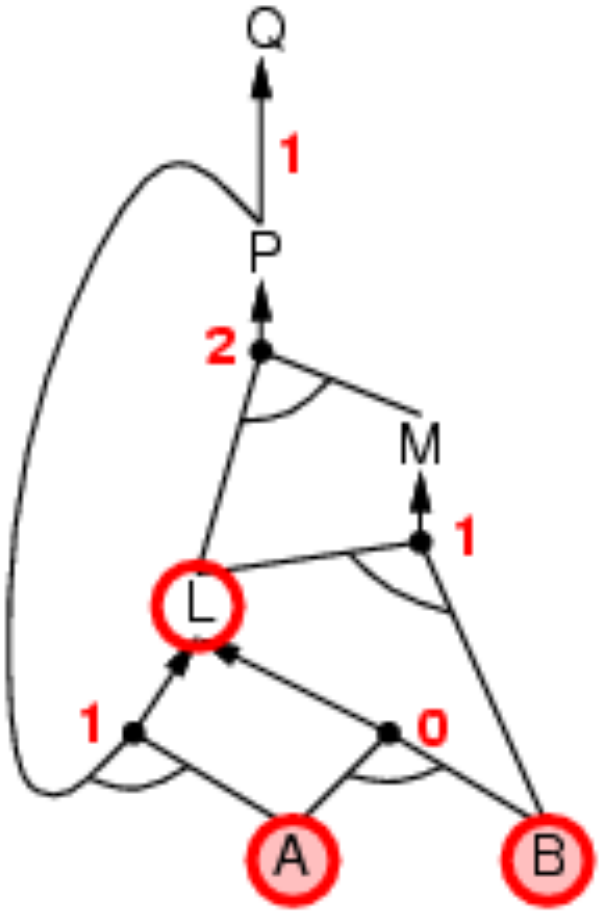
Forward chaining: esempio



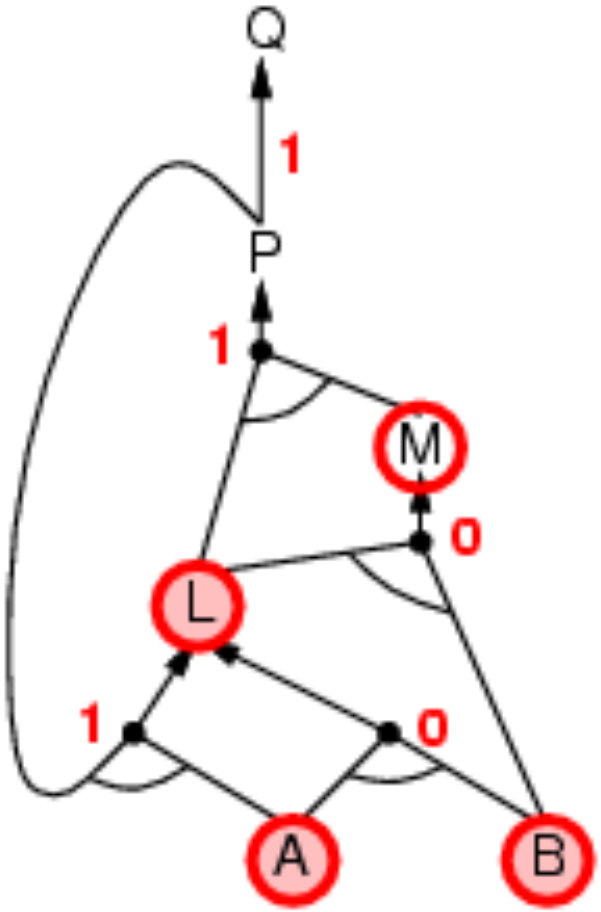
Forward chaining : esempio



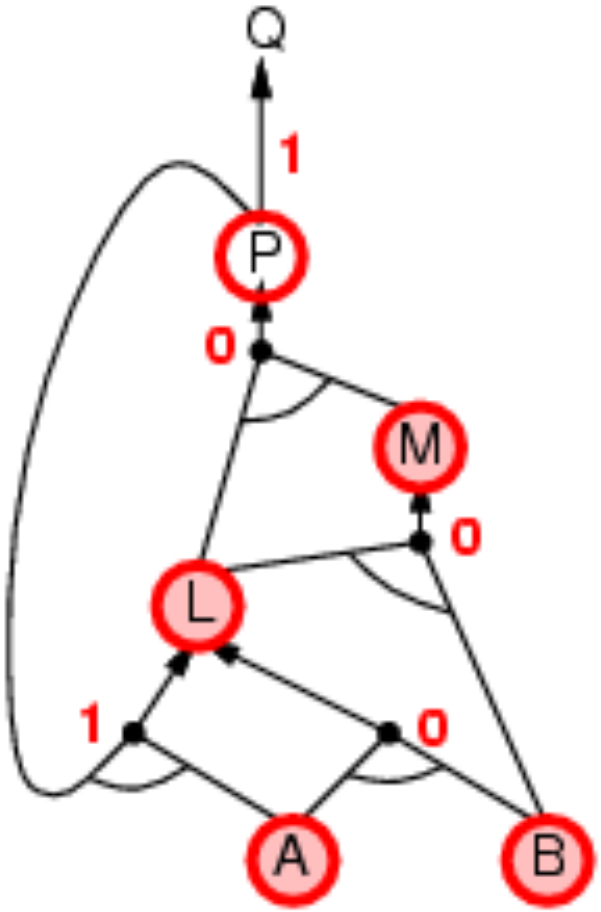
Forward chaining : esempio



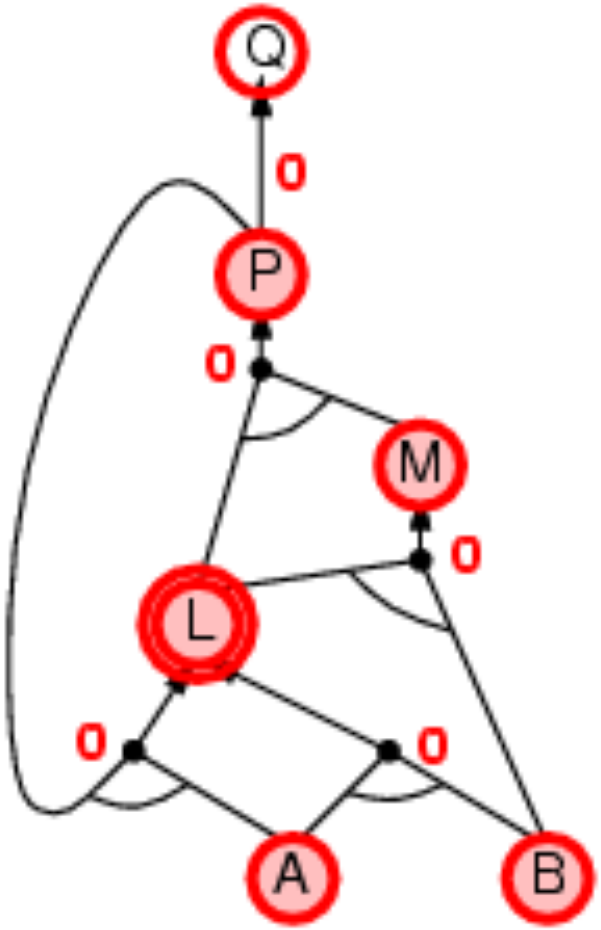
Forward chaining : esempio



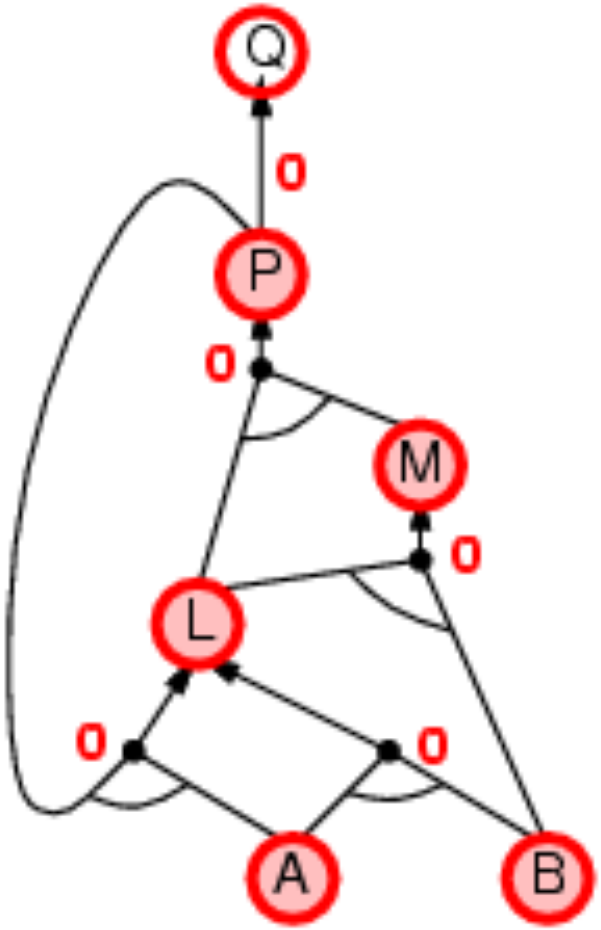
Forward chaining : esempio



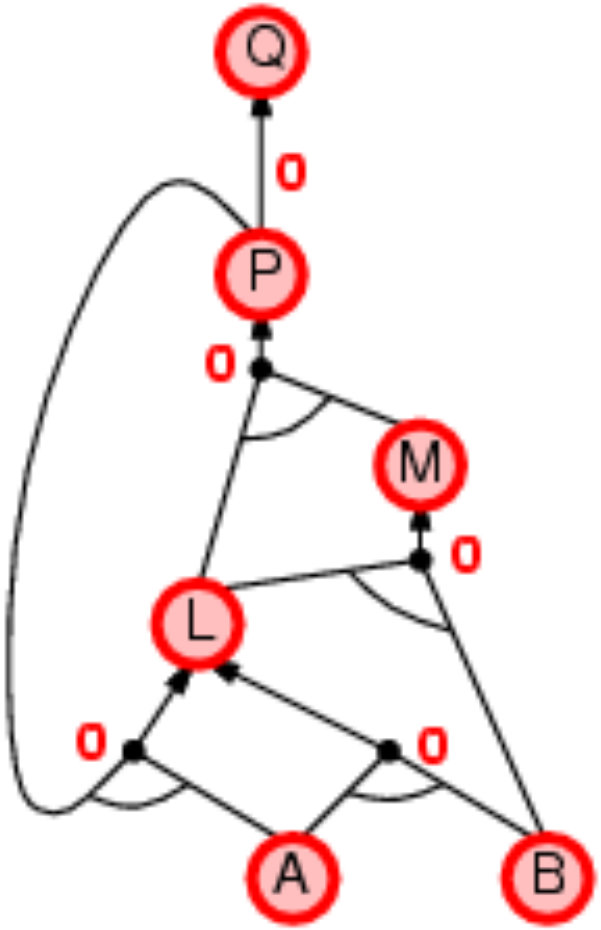
Forward chaining : esempio



Forward chaining : esempio



Forward chaining : esempio



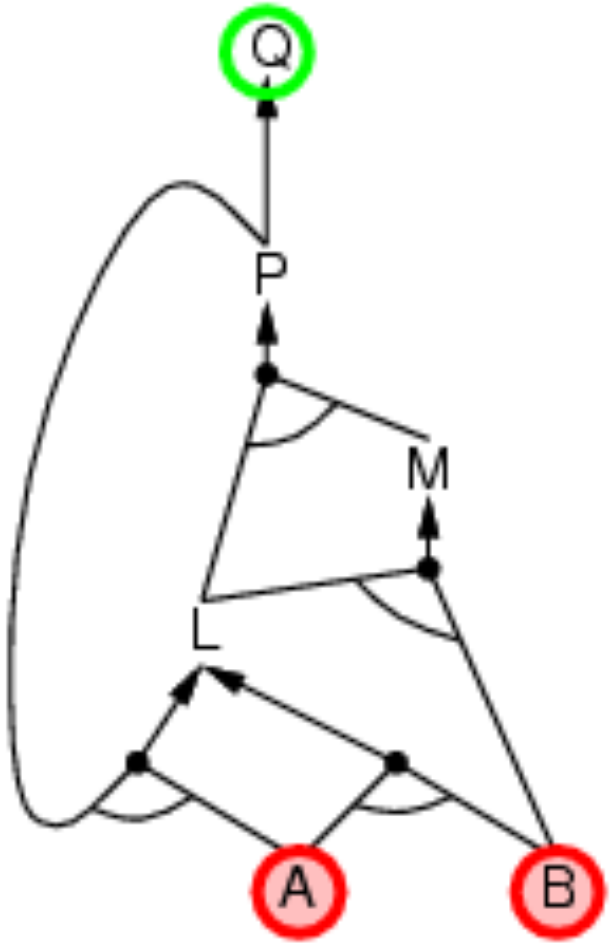
Backward chaining (BC)

- Idea: lavora *backward* dalla query (goal) *q*:
  - Per dimostrare *q*
    - Controlla se *q* è gia noto, oppure,
    - Dimostra attraverso BC tutte le premesse di una qualche regola che conclude *q*

- Per evitare *loops*: controlla se i nuovi sottogoal sono già nello stack dei goal

- Evita di ripetere lavoro controllando se i nuovi sottogoal sono già stati dimostrati veri o falsi.

Backward chaining : esempio



$P \Rightarrow Q$

$L \wedge M \Rightarrow P$

$B \wedge L \Rightarrow M$

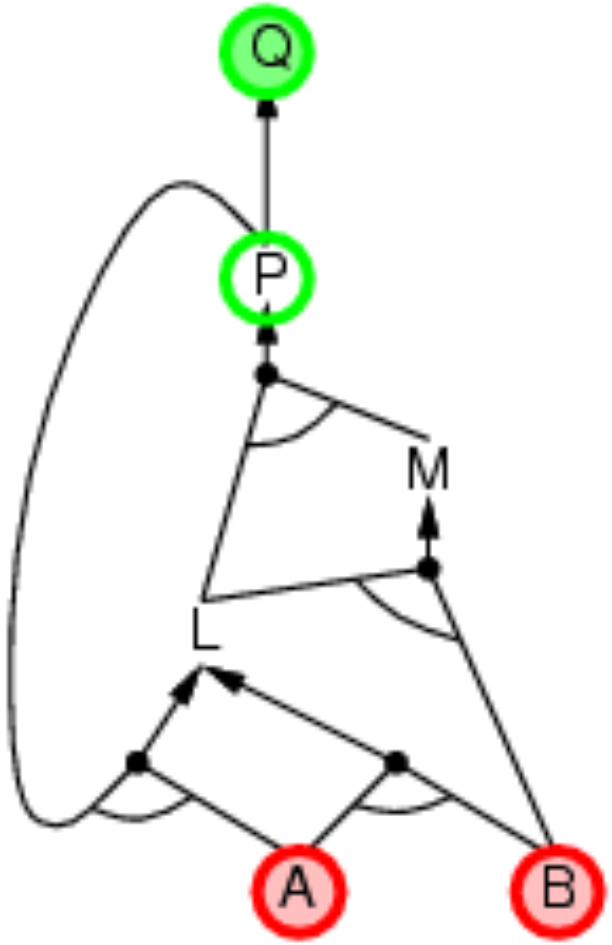
$A \wedge P \Rightarrow L$

$A \wedge B \Rightarrow L$

A

B

Backward chaining : esempio



$P \Rightarrow Q$

$L \wedge M \Rightarrow P$

$B \wedge L \Rightarrow M$

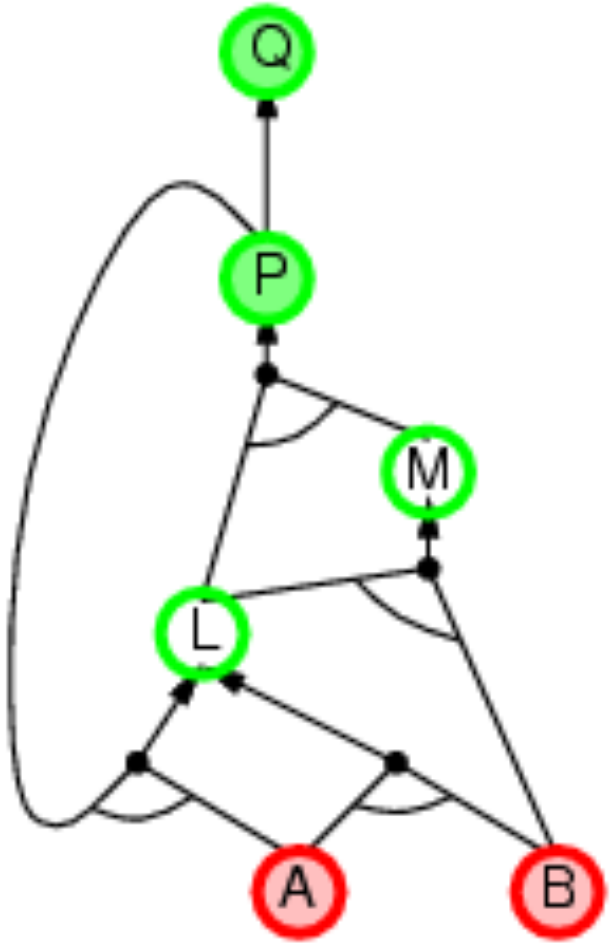
$A \wedge P \Rightarrow L$

$A \wedge B \Rightarrow L$

A

B

Backward chaining : esempio



$P \Rightarrow Q$

$L \wedge M \Rightarrow P$

$B \wedge L \Rightarrow M$

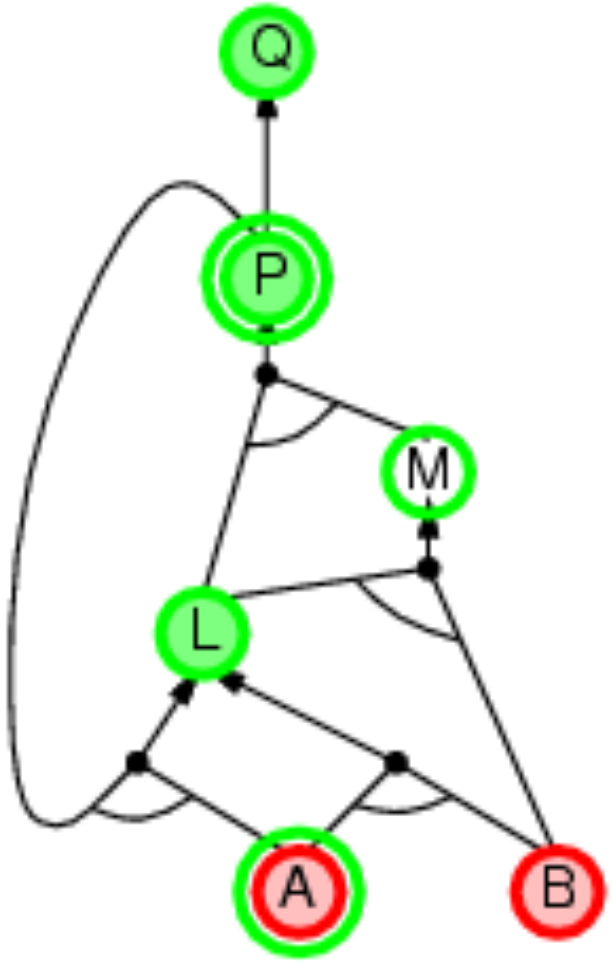
$A \wedge P \Rightarrow L$

$A \wedge B \Rightarrow L$

A

B

Backward chaining : esempio



$P \Rightarrow Q$

$L \wedge M \Rightarrow P$

$B \wedge L \Rightarrow M$

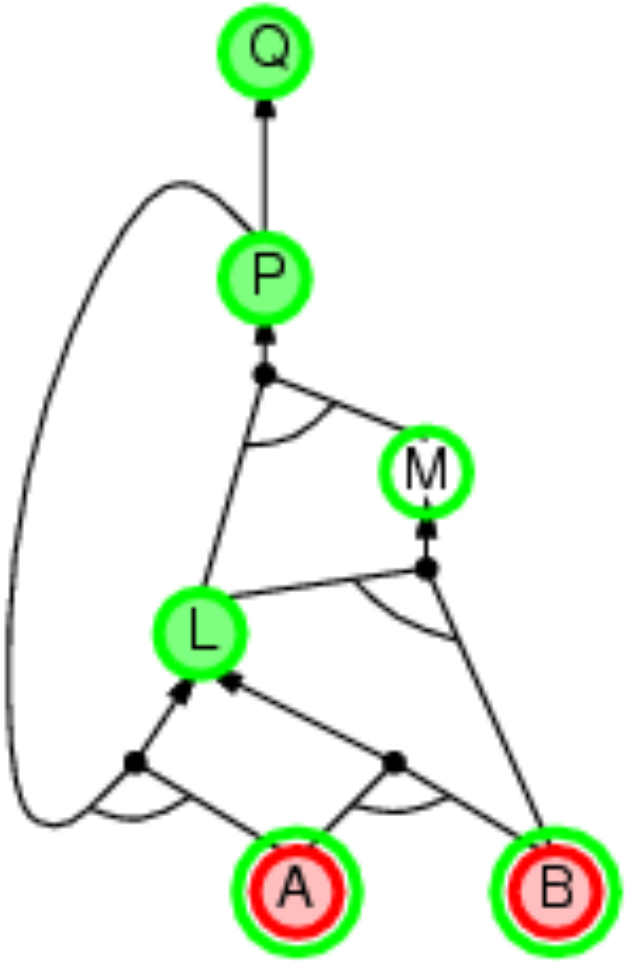
$A \wedge P \Rightarrow L$

$A \wedge B \Rightarrow L$

A

B

Backward chaining : esempio



$P \Rightarrow Q$

$L \wedge M \Rightarrow P$

$B \wedge L \Rightarrow M$

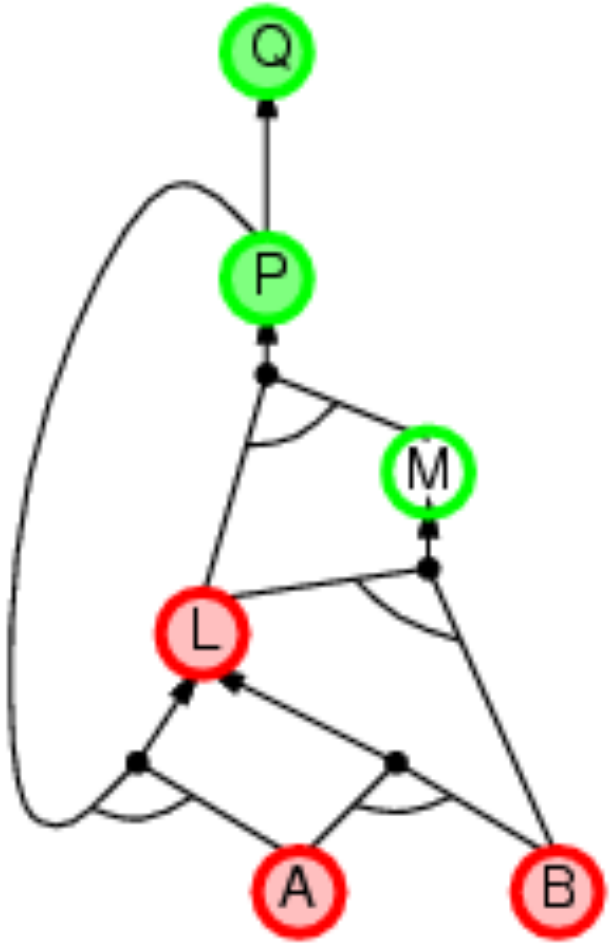
$A \wedge P \Rightarrow L$

$A \wedge B \Rightarrow L$

A

B

Backward chaining : esempio



$P \Rightarrow Q$

$L \wedge M \Rightarrow P$

$B \wedge L \Rightarrow M$

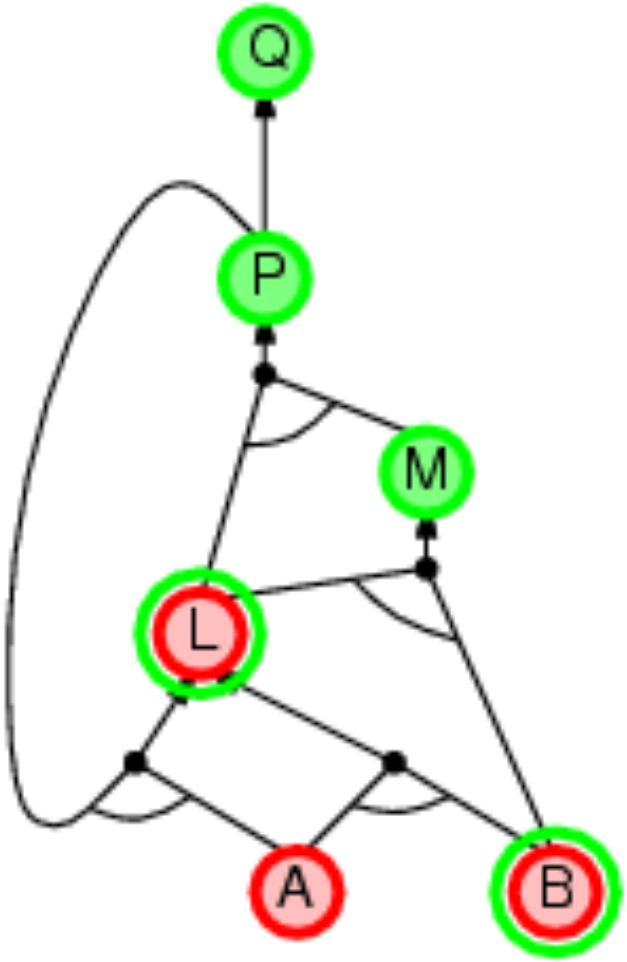
$A \wedge P \Rightarrow L$

$A \wedge B \Rightarrow L$

A

B

Backward chaining : esempio



$P \Rightarrow Q$

$L \wedge M \Rightarrow P$

$B \wedge L \Rightarrow M$

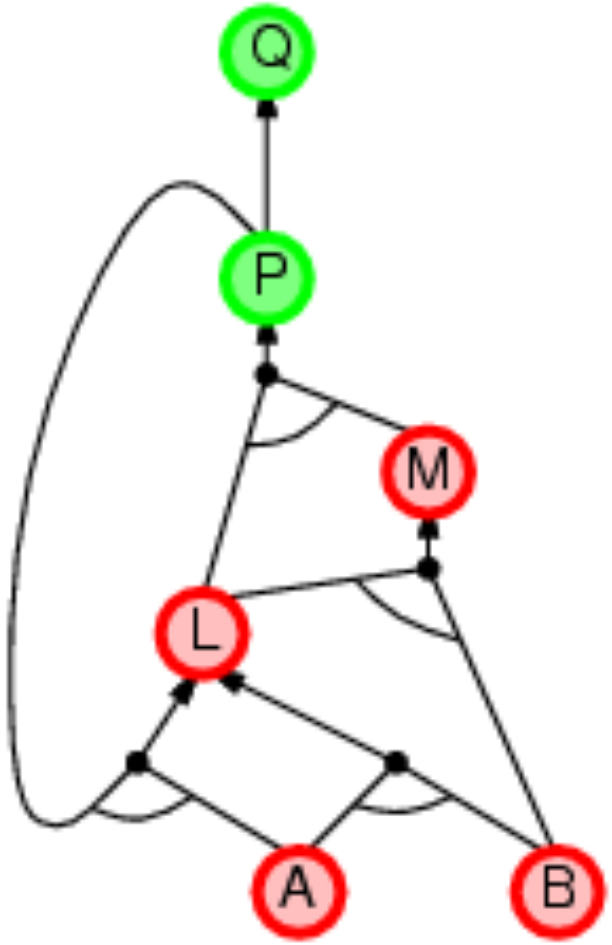
$A \wedge P \Rightarrow L$

$A \wedge B \Rightarrow L$

A

B

Backward chaining : esempio



$P \Rightarrow Q$

$L \wedge M \Rightarrow P$

$B \wedge L \Rightarrow M$

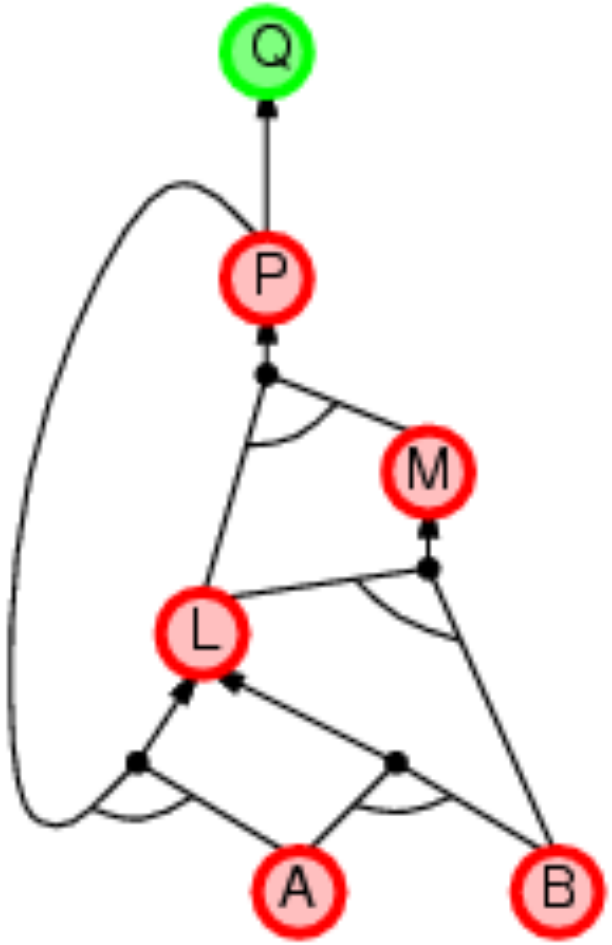
$A \wedge P \Rightarrow L$

$A \wedge B \Rightarrow L$

A

B

Backward chaining : esempio



$P \Rightarrow Q$

$L \wedge M \Rightarrow P$

$B \wedge L \Rightarrow M$

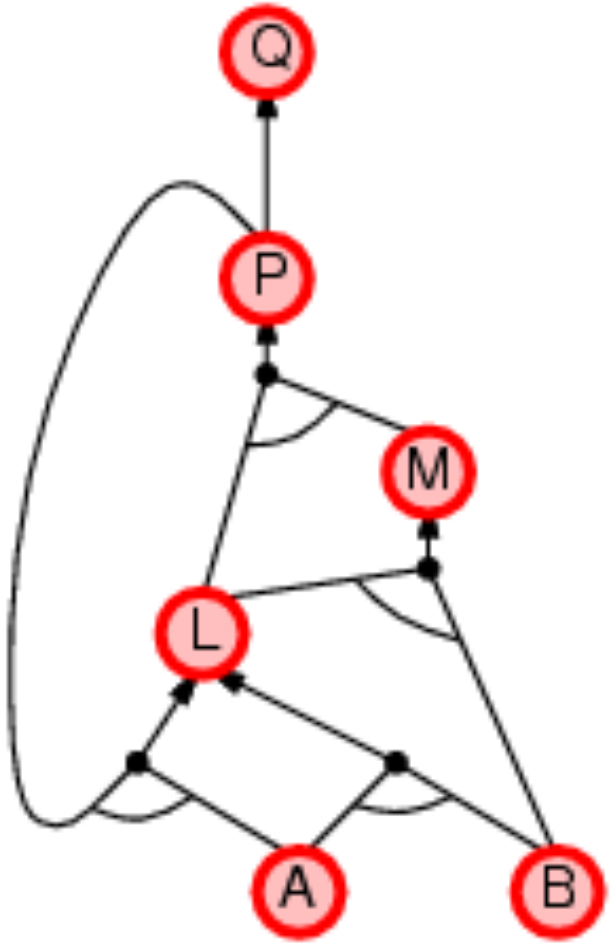
$A \wedge P \Rightarrow L$

$A \wedge B \Rightarrow L$

A

B

Backward chaining : esempio



$P \Rightarrow Q$

$L \wedge M \Rightarrow P$

$B \wedge L \Rightarrow M$

$A \wedge P \Rightarrow L$

$A \wedge B \Rightarrow L$

A

B

Forward vs. backward chaining

- FC è data-driven,
  - e.g., monitoring, configurazione, interpretazione
  - Non si concentra sull'obiettivo
- BC è goal-driven, quindi in generale più appropriato per il problem-solving

Sommario (1)

- Risoluzione , opera su clausole

$$\frac{A_1 \vee \ldots \vee A_n \qquad B_1 \vee \ldots \vee B_m \qquad A_i = \sim B_j}{A_1 \vee .. \vee A_{i-1} \vee A_{i+1} \vee .. \vee A_n \vee B_1 \vee .. \vee B_{j-1} \vee B_{j+1} \vee \ldots \vee B_m}$$

- Forward, backward chaining su clausole di Horn

$$\frac{\alpha_1, \ldots, \alpha_n, \qquad \alpha_1 \wedge \ldots \wedge \alpha_n \Rightarrow \beta}{\beta}$$

- Goal driven (BC) *vs* data driven (FC)

Sommario (2)

- Risoluzione

$$\frac{A_1 \vee \ldots \vee A_n \qquad B_1 \vee \ldots \vee B_m \qquad A_i = \sim B_j}{A_1 \vee \ldots \vee A_{i-1} \vee A_{i+1} \vee \ldots \vee A_n \vee B_1 \vee \ldots \vee B_{j-1} \vee B_{j+1} \vee \ldots \vee B_m}$$

- Forward chaining (MP, applicato n volte)

$$\frac{\alpha_1 \ \ldots \ \alpha_n \qquad \sim \alpha_1 \vee \ldots \vee \sim \alpha_n \vee \beta}{\beta}$$

- Backward chaining (in refutation mode)

$$\frac{\sim\beta \qquad \sim(\alpha_1 \wedge \ldots \wedge \alpha_n) \vee \beta}{\sim(\alpha_1 \wedge \ldots \wedge \alpha_n)}$$

Backward chaining, per refutazione

$P \Rightarrow Q$

$L \wedge M \Rightarrow P$

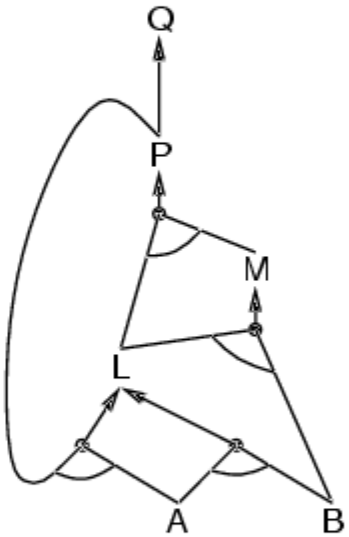
$B \wedge L \Rightarrow M$

$A \wedge P \Rightarrow L$

$A \wedge B \Rightarrow L$

A

B



$\sim Q$ *(query negata)*

$\sim P$

$\sim(L \wedge M) \equiv \sim L \vee \sim M$

$\sim(A \wedge B) \vee \sim M \equiv \sim A \vee \sim B \vee \sim M$

$\sim B \vee \sim M$

$\sim M$

$\sim(B \wedge L) \equiv \sim B \vee \sim L$

$\sim L$

$\sim(A \wedge B) \equiv \sim A \vee \sim B$

$\sim B$

clausola vuota

Sommario (3)

- La risoluzione è completa per la logica proposizionale
- Forward, backward chaining sono complete per le clausole di Horn
- Risoluzione e backward chaining usate nei sistemi che operano per refutazione

- Però la logica proposizionale è troppo povera dal punto di vista espressivo.