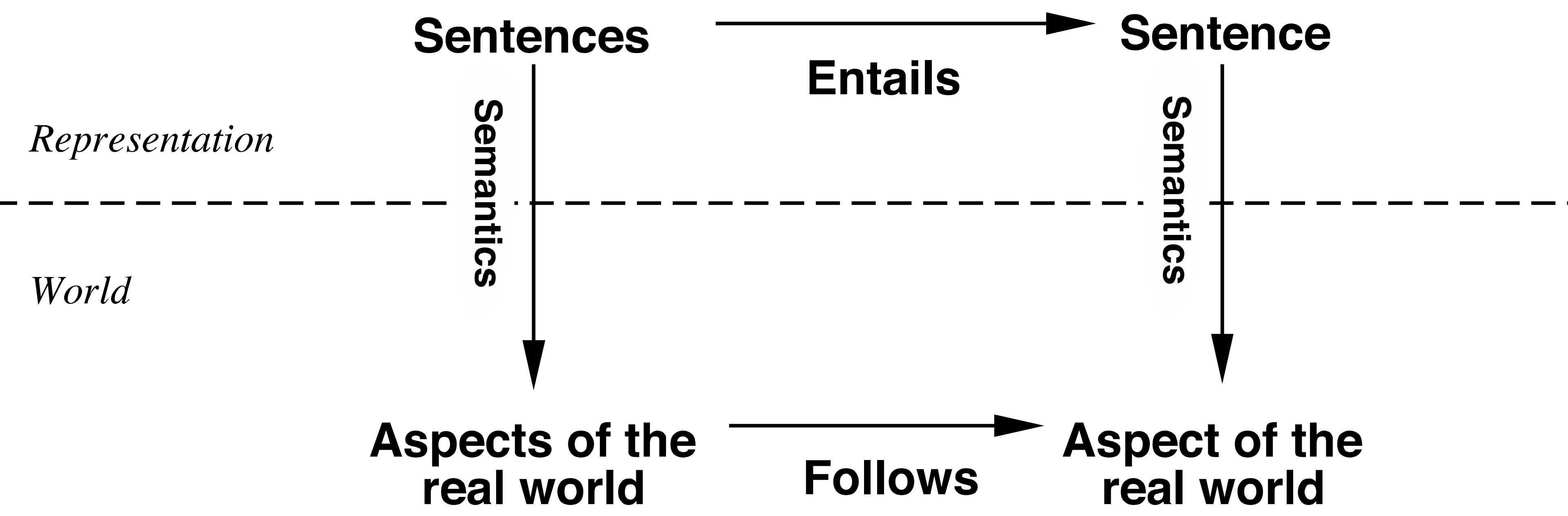
Rappresentazione e elaborazione della conoscenza: dalla sintassi (frasi della KB) alla semantica (mondo reale)



Entails: implica
Follows: conseguenza (logica)

Procedura di Inferenza/Deduzione

- "*Metodo automatico per derivare nuove frasi (formule) a partire da quelle già presenti nella base di conoscenza (KB)*"

- **Corretta** ("sound") se per ogni formula ϕ derivata da KB usando la procedura di inferenza si ha che ϕ e' una *conseguenza logica* di KB. **Completa** se vale anche il viceversa.

- **Dimostrazione** di ϕ: insieme di operazioni eseguite dalla procedura di inferenza (corretta) per derivare ϕ

- **Teoria della dimostrazione** ("proof theory"): è relativa ad un longuaggio di KR. Specifica un insieme di passi di ragionamento/inferenza che sono corretti

Alcuni Linguaggi di KR (Knowledge Representation)

- Un linguaggio di KR deve essere **conciso**, **espressivo** (il più possibile), **non ambiguo** e indipendente dal contesto, **efficace** (esiste una procedura di inferenza corretta ed implementabile)

- **Linguaggio naturale**: espressivo, ma ambiguo, non conciso

- **Linguaggio di programmazione**: preciso, strutturato, ma poco espressivo

- **Logica**: precisa, concisa, expressiva

Logica e KR

- Varie logiche, differenti assunzioni ontologiche e epistemologiche (esempi)
- Una logica è descritta da *sintassi* e *semantica*
- Vedremo Logica Proposizionale e Logica del I Ordine (Calcolo dei Predicati) in dettaglio, Logica Temporal (cenni)

Assunzioni Ontologiche/Epistemiologiche nei Linguaggi di KR

Language	Ontological Commitment (What exists in the world)	Epistemological Commitment (What an agent believes about facts)
Propositional logic	facts	true/false/unknown
First-order logic	facts, objects, relations	true/false/unknown
Temporal logic	facts, objects, relations, times	true/false/unknown
Probability theory	facts	degree of belief $\in [0, 1]$
Fuzzy logic	facts with degree of truth $\in [0, 1]$	known interval value

Figure 8.1 Formal languages and their ontological and epistemological commitments.

Sintassi logica proposizionale

$$
\begin{array}{rcl}
Sentence & \rightarrow & AtomicSentence \mid ComplexSentence \\
AtomicSentence & \rightarrow & True \mid False \mid P \mid Q \mid R \mid \ldots \\
ComplexSentence & \rightarrow & (\ Sentence\ ) \mid [\ Sentence\ ] \\
& \mid & \neg\ Sentence \\
& \mid & Sentence \land Sentence \\
& \mid & Sentence \lor Sentence \\
& \mid & Sentence \ \Rightarrow\ Sentence \\
& \mid & Sentence \ \Leftrightarrow\ Sentence \\
\\
\text{OPERATOR PRECEDENCE} & : & \neg, \land, \lor, \Rightarrow, \Leftrightarrow
\end{array}
$$

Figure 7.7 A BNF (Backus–Naur Form) grammar of sentences in propositional logic, along with operator precedences, from highest to lowest.

Semantica operatori logici

P	Q	$\neg P$	$P \wedge Q$	$P \vee Q$	$P \Rightarrow Q$	$P \Leftrightarrow Q$
false	*false*	*true*	*false*	*false*	*true*	*true*
false	*true*	*true*	*false*	*true*	*true*	*false*
true	*false*	*false*	*false*	*true*	*false*	*false*
true	*true*	*false*	*true*	*true*	*true*	*true*

Figure 7.8 Truth tables for the five logical connectives. To use the table to compute, for example, the value of $P \vee Q$ when P is true and Q is false, first look on the left for the row where P is *true* and Q is *false* (the third row). Then look in that row under the $P \vee Q$ column to see the result: *true*.

Semantica della Logica

- Ciò che la formula asserisce sul mondo = **significato della frase**
- Dipende da una particolare **interpretazione**: Una formula è **vera rispetto ad una interpretazione** se ciò che rappresenta è vero nel mondo.

  Es: “*il Papa è in Francia*” ... → *“la chiave è nel frigorifero”*

- **Principio di composizionalità**: il significato di una formula è una funzione del significato delle sue parti (si per la logica!)
- **Formula valida (tautologia):** vera per qualsiasi interpretazione (“*piove **o** non piove*”. Altri esempi meno ovvi dopo
- **Formula soddisfacibile**: esiste una interpretazione per cui la formula è vera (“*piove **e** fa-caldo*”), **insoddifacibile** (“*piove **e** non piove*”) altrimenti
- Ogni formula tautologica è soddisfacibile (no vicerersa)

Modelli (logica proposizionale)

- Un **Modello** per una (formula ben formata) F.B.F ϕ è una intepretazione (associazione) dei simboli proposizionali di ϕ in valori di verità (V,F) per i quali ϕ è vero.

- Modello per ϕ: "mondo" in cui ϕ è vero rispetto ad una certa interpretazione

- Esempio, modelli di "P $\Rightarrow$ not Q": [F,F], [F,V], [V,F]

- Servono per definire il concetto (**semantico**) di **conseguenza logica**: <u>Se l'insieme di tutti i modelli di una KB di F.B.F. sono anche modelli di ϕ, allora ϕ è una conseguenza logica di KB</u>

KB = {"piove", "piove $\Rightarrow$ bagnato"} , ϕ = "bagnato"

Validity and Unsatisfiability

Validità

A sentence is **valid** if it is true in **all** interpretatioms,

Ad es. *True*, $A \vee \neg A$, $A \Rightarrow A$, $(A \wedge (A \Rightarrow B)) \Rightarrow B$

Validity is connected to inference via the **Deduction Theorem**:

$KB \models \alpha$ if and only if $(KB \Rightarrow \alpha)$ is valid

Insoddisfacibilità

A sentence is **unsatisfiable** if it is true in **no** interpretation

e.g., $A \wedge \neg A$

Logical implication (entailment) is connected to unsatisfiability via the following: $\boldsymbol{KB \models \alpha}$ $((KB \Rightarrow \alpha)$ is valid)

if and only if $(KB \wedge \neg\alpha)$ is unsatisfiable

Metodi semantici (model checking)

- Non usano procedure di inferenza, ma modelli (dimostriamo la conseguenza logica).

- Metodo basato su **tabelle di verità** (esempi)

KB = {A, not B, (A or B $\Rightarrow$ C)} ***KB*** $\models$ ***C ?***

KB = {A, not B, (A or B $\Rightarrow$ C)} ***KB*** $\models$ ***(B*** $\Rightarrow$ ***C) ?***

- Complessità temporale e spaziale *esponenziale*.....

$KB=\{A, \text{not } B, (A \text{ or } B \Rightarrow C)\}$ $\boldsymbol{KB \models C}$?

$KB=\{A, \text{not } B, (A \text{ or } B \Rightarrow C)\}$ $\boldsymbol{KB} \models B \Rightarrow C$?

A	**B**	**C**	**not B**	**A or B ⇒ C**	**B ⇒ C**
V	V	V	F	V	V
V	V	F	F	F	F
V	F	V	V	V	V
V	F	F	V	F	V
F	V	V	F	V	V
F	V	F	F	F	F
F	F	V	V	V	V
F	F	F	V	V	V

1 MODELLO DI KB: A=V, B=F, C=V

Altro Metodo Semantico

Φ *è una conseguenza logica di una KB consistente (soddisfacibile) se e solo se KB' = KB* $\wedge$ *NOT* Φ *è insoddifacibile (non ha modelli).*

- Formulo il problema come **CSP Booleano** (**SAT**) !! Se trovo un modello per KB', allora è soddisfacibile e Φ non è una conseguenza logica di KB (altrimenti lo è)
- Complessità esponenziale nel caso generale, polinomiale per 2SAT e per *formule di Horn*

Dimostrazioni e Inferenza (ragionamento deduttivo)

- **Dimostrazione (derivazione) logica** di una formula **f**: sequenza di applicazioni di ***regole di inferenza*** a partire dalle F.B.F. nella KB fino a generare il **teorema** **f**
- **Procedura di inferenza**: procedura per costruire dimostrazioni logiche attraverso l'individuazione di appropriate *sequenze di applicazioni di regole di inferenza*
- **Regole di inferenza**: *Modus ponens, and-elimination, and-introduction, or-introduction, double-negation, unit resolution, resolution*
- Richiedono formule in *formato normalizzato*
- Esempi di dimostrazione

Resolution

Conjunctive Normal Form (CNF)
= conjunction of disjunctions of literals (= conjunction of clauses)

Example: $(A \vee \neg B)$, $(B \vee \neg C \vee \neg D)$

- Resolution inference rule (for CNF):

$$\frac{\ell_1 \vee \ldots \vee \ell_k, \qquad m_1 \vee \ldots \vee m_n}{\ell_1 \vee \ldots \vee \ell_{i-1} \vee \ell_{i+1} \vee \ldots \vee \ell_k \vee m_1 \vee \ldots \vee m_{j-1} \vee m_{j+1} \vee \ldots \vee m_n}$$

where ℓ_i and m_j are *complementary* literals.

Resolution is sound and complete for propositional logic

Resolution

Implicative version of the rule

$$\frac{\neg(l_i \vee \ldots \vee l_{i-1} \vee l_{i+1} \vee \ldots \vee l_k) \Rightarrow l_i \qquad \neg m_j \Rightarrow (m_1 \vee \ldots \vee m_{j-1} \vee m_{j+1} \vee \ldots \vee m_n)}{\neg(l_i \vee \ldots \vee l_{i-1} \vee l_{i+1} \vee \ldots \vee l_k) \Rightarrow (m_1 \vee \ldots \vee m_{j-1} \vee m_{j+1} \vee \ldots \vee m_n)}$$

Conversion to CNF

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

1. Eliminate $\Leftrightarrow$, replacing $\alpha \Leftrightarrow \beta$ with $(\alpha \Rightarrow \beta) \land (\beta \Rightarrow \alpha)$.
   $(B_{1,1} \Rightarrow (P_{1,2} \lor P_{2,1})) \land ((P_{1,2} \lor P_{2,1}) \Rightarrow B_{1,1})$

2. Eliminate $\Rightarrow$, replacing $\alpha \Rightarrow \beta$ with $\neg\alpha \lor \beta$.

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

3. Move $\neg$ inwards using de Morgan's rules and double-negation:

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

4. Apply distributivity law ($\land$ over $\lor$) and flatten:

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

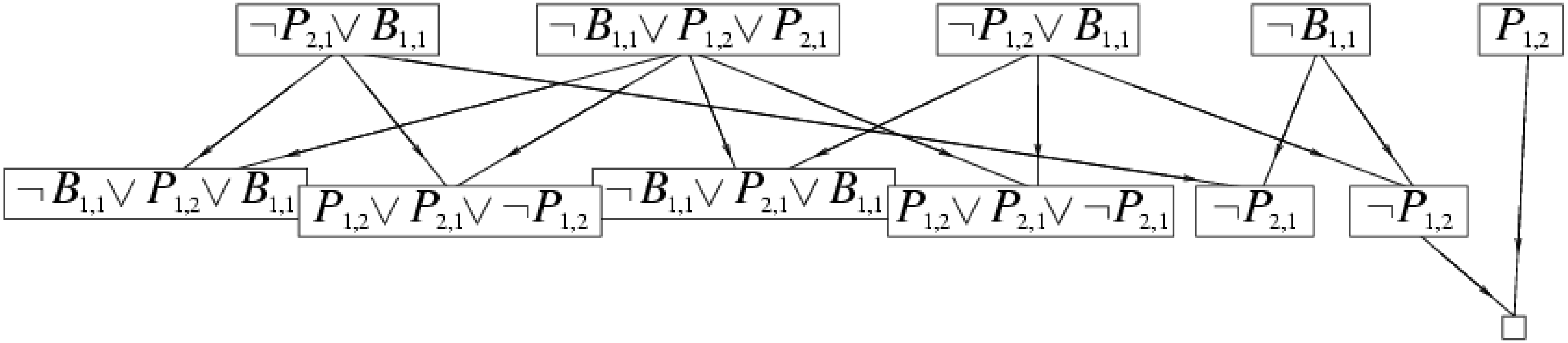
Some Logical Equivalences

$$
\begin{aligned}
(\alpha \wedge \beta) &\equiv (\beta \wedge \alpha) \quad \text{commutativity of } \wedge \\
(\alpha \vee \beta) &\equiv (\beta \vee \alpha) \quad \text{commutativity of } \vee \\
((\alpha \wedge \beta) \wedge \gamma) &\equiv (\alpha \wedge (\beta \wedge \gamma)) \quad \text{associativity of } \wedge \\
((\alpha \vee \beta) \vee \gamma) &\equiv (\alpha \vee (\beta \vee \gamma)) \quad \text{associativity of } \vee \\
\neg(\neg \alpha) &\equiv \alpha \quad \text{double-negation elimination} \\
(\alpha \Rightarrow \beta) &\equiv (\neg \beta \Rightarrow \neg \alpha) \quad \text{contraposition} \\
(\alpha \Rightarrow \beta) &\equiv (\neg \alpha \vee \beta) \quad \text{implication elimination} \\
(\alpha \Leftrightarrow \beta) &\equiv ((\alpha \Rightarrow \beta) \wedge (\beta \Rightarrow \alpha)) \quad \text{biconditional elimination} \\
\neg(\alpha \wedge \beta) &\equiv (\neg \alpha \vee \neg \beta) \quad \text{De Morgan} \\
\neg(\alpha \vee \beta) &\equiv (\neg \alpha \wedge \neg \beta) \quad \text{De Morgan} \\
(\alpha \wedge (\beta \vee \gamma)) &\equiv ((\alpha \wedge \beta) \vee (\alpha \wedge \gamma)) \quad \text{distributivity of } \wedge \text{ over } \vee \\
(\alpha \vee (\beta \wedge \gamma)) &\equiv ((\alpha \vee \beta) \wedge (\alpha \vee \gamma)) \quad \text{distributivity of } \vee \text{ over } \wedge
\end{aligned}
$$

Figure 7.11 Standard logical equivalences. The symbols α, β, and γ stand for arbitrary sentences of propositional logic.

Resolution example

- $KB = (B_{1,1} \Leftrightarrow (P_{1,2} \lor P_{2,1})) \land \neg B_{1,1}$
- $\alpha = \neg P_{1,2}$
-



Completezza della risoluzione

- *Una procedura di inferenza che applica la regola di risoluzione con una* ***opportuna strategia*** *è* ***completa*** *per dimostrare, attraverso il* ***metodo di refutazione, qualsiasi formula*** *della logica proposizionale*

- Esempio: Se KB = {A} non posso derivare "A or B" con la risoluzione, ma posso dimostrare "A or B" attraverso il metodo di refutazione….

- **Strategia di risoluzione =** come scelgo le 2 clausole da risolvere

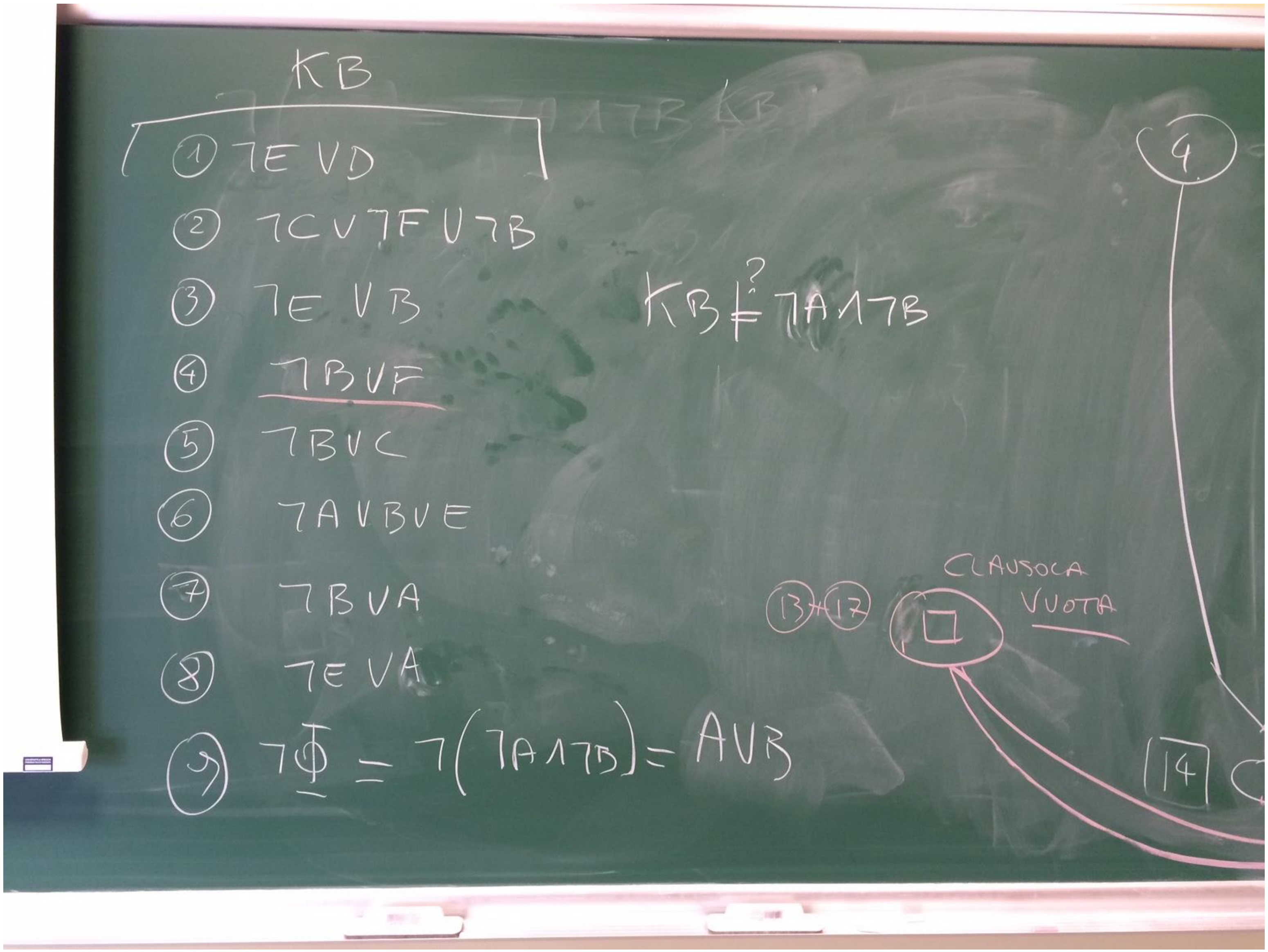
- La KB congiunta con la formula negata devono essere in **formato CNF** (congiunzione/insieme di clausole)

Resolution algorithm

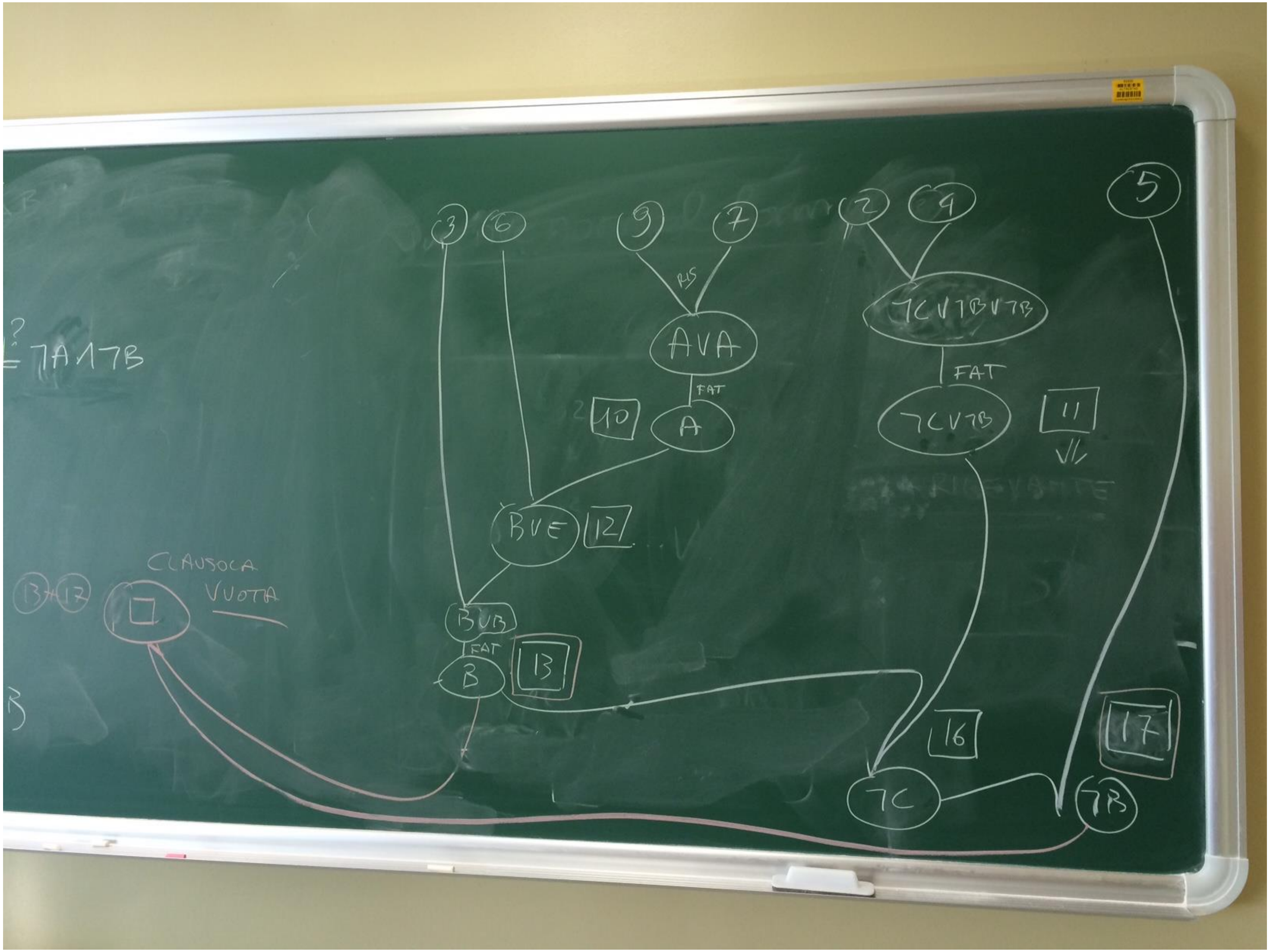
- Proof by contradiction, i.e., show $KB \wedge \neg \alpha$ unsatisfiable

```
function PL-RESOLUTION(KB, α) returns true or false
    clauses ← the set of clauses in the CNF representation of KB ∧ ¬α
    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
```

Resolution example (2)



Solution example 2 (refutation graph)



CNF, Horn, Clausole Definite

$$
\begin{aligned}
CNFSentence &\rightarrow Clause_1 \wedge \cdots \wedge Clause_n \\
Clause &\rightarrow Literal_1 \vee \cdots \vee Literal_m \\
Literal &\rightarrow Symbol \mid \neg Symbol \\
Symbol &\rightarrow P \mid Q \mid R \mid \ldots \\
HornClauseForm &\rightarrow DefiniteClauseForm \mid GoalClauseForm \\
DefiniteClauseForm &\rightarrow (Symbol_1 \wedge \cdots \wedge Symbol_l) \Rightarrow Symbol \\
GoalClauseForm &\rightarrow (Symbol_1 \wedge \cdots \wedge Symbol_l) \Rightarrow False
\end{aligned}
$$

Figure 7.14 A grammar for conjunctive normal form, Horn clauses, and definite clauses. A clause such as $A \wedge B \Rightarrow C$ is still a definite clause when it is written as $\neg A \vee \neg B \vee C$, but only the former is considered the canonical form for definite clauses. One more class is the *k*-CNF sentence, which is a CNF sentence where each clause has at most *k* literals.

Forward and backward chaining

- Horn Form (restricted)

  KB = conjunction of Horn clauses (DEFINITE CLAUSES)

  - Horn clause =
    - proposition symbol; or
    - (conjunction of symbols) $\Rightarrow$ symbol
  - E.g., $C \wedge (B \Rightarrow A) \wedge (C \wedge D \Rightarrow B)$

- Modus Ponens (for Horn Form): complete for Horn KBs

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

- Can be used with forward chaining or backward chaining.
- These algorithms are very natural and run in **linear** time!

Esercizi

$\neg P \vee Q$

$\neg L \vee \neg M \vee P$

$\neg B \vee \neg L \vee M$

$\neg A \vee \neg P \vee L$

$\neg A \vee \neg B \vee L$

A

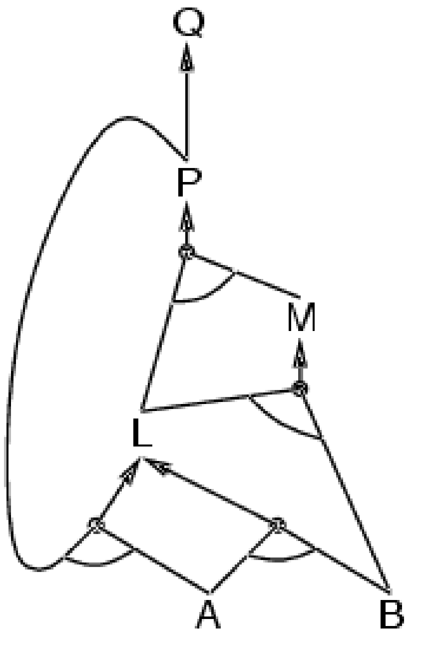
B

1. Sono clausole di Horn? Sono definite?
2. Convertite in forma implicative (come vuole modus ponens)
3. Costruire l'ipergrafo AND-OR che rappresenta KB
4. Dimostrare se $KB \models Q$ usando
   - risoluzione con strategie: **ampiezza**, clausola **unitaria** positiva, **lineare**
   - forward chaining (dopo trasformazione in clausole definite implicative)

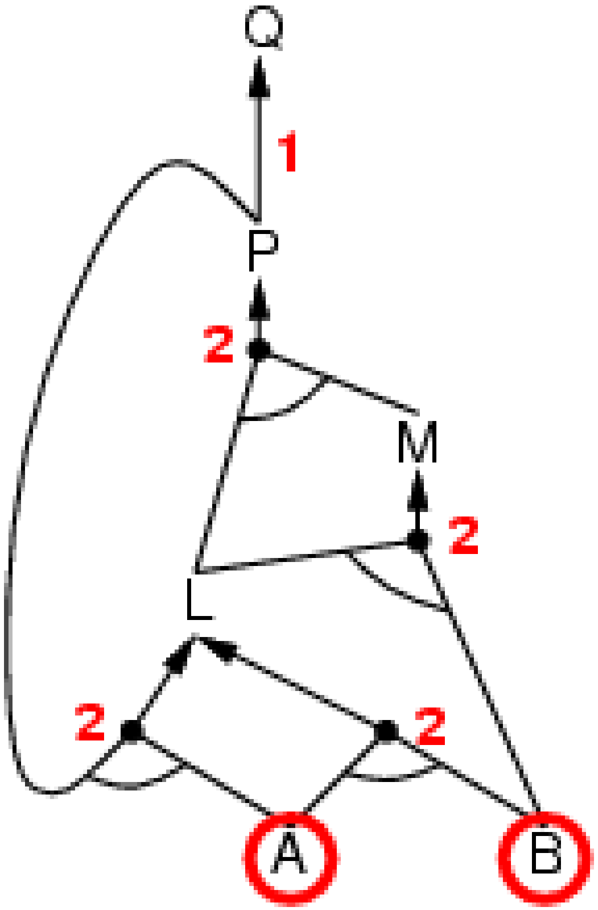
Forward chaining

- Idea: fire any rule whose premises are satisfied in the *KB*,
  - add its conclusion to the *KB*, until query is found

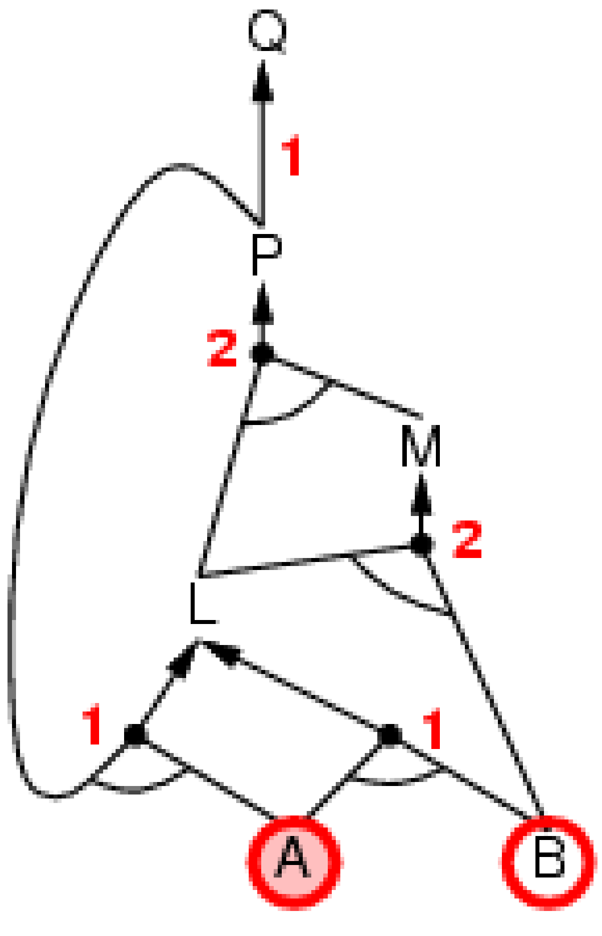
$$P \Rightarrow Q$$
$$L \wedge M \Rightarrow P$$
$$B \wedge L \Rightarrow M$$
$$A \wedge P \Rightarrow L$$
$$A \wedge B \Rightarrow L$$
$$A$$
$$B$$



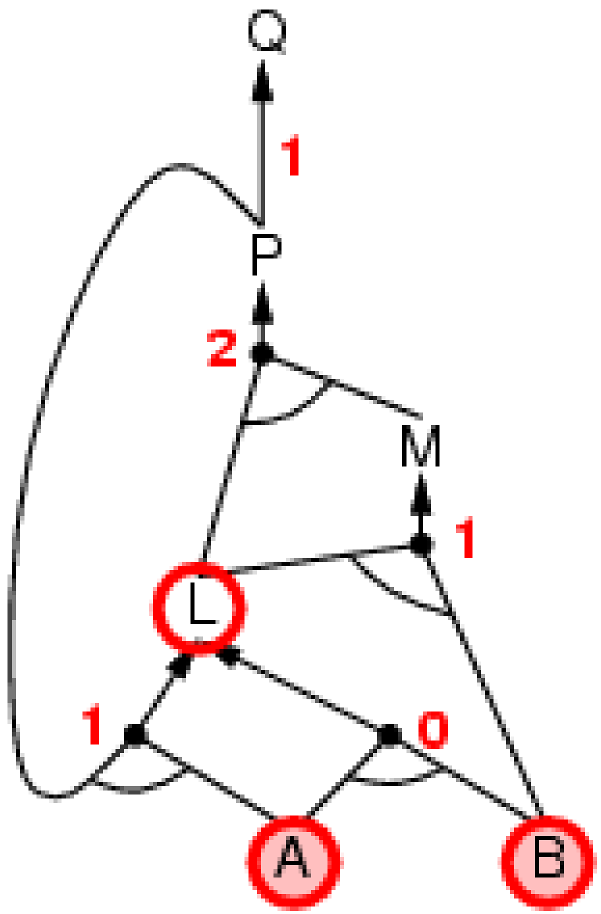
Forward chaining example



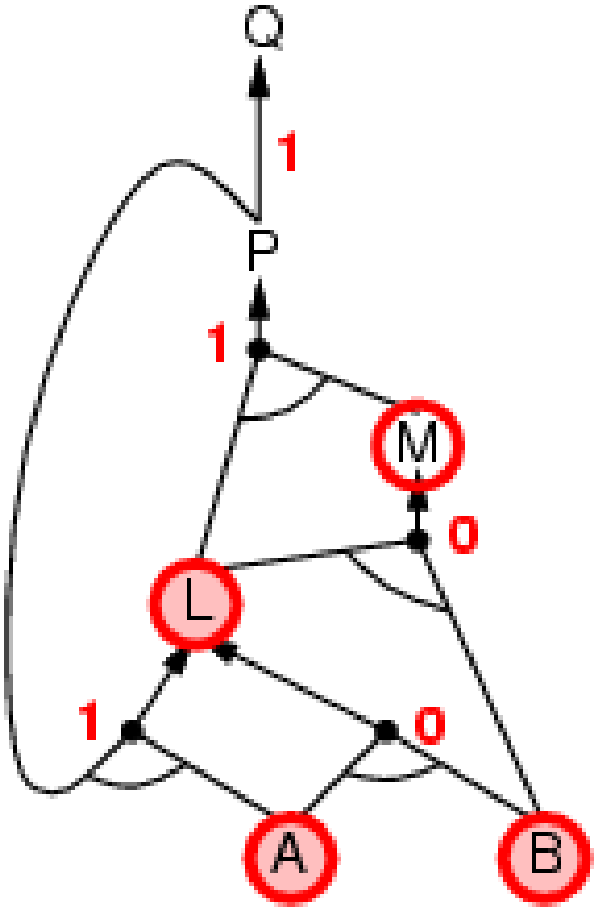
Forward chaining example



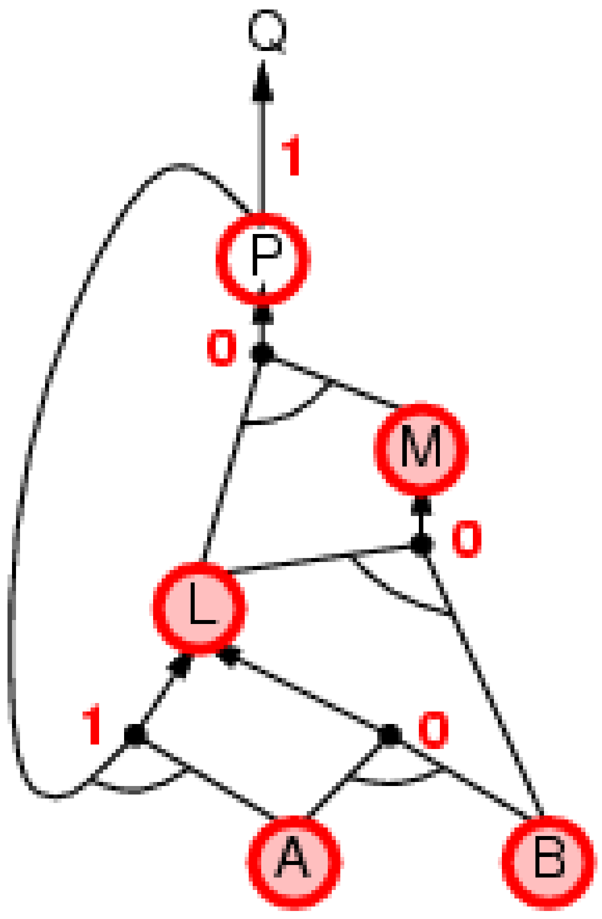
Forward chaining example



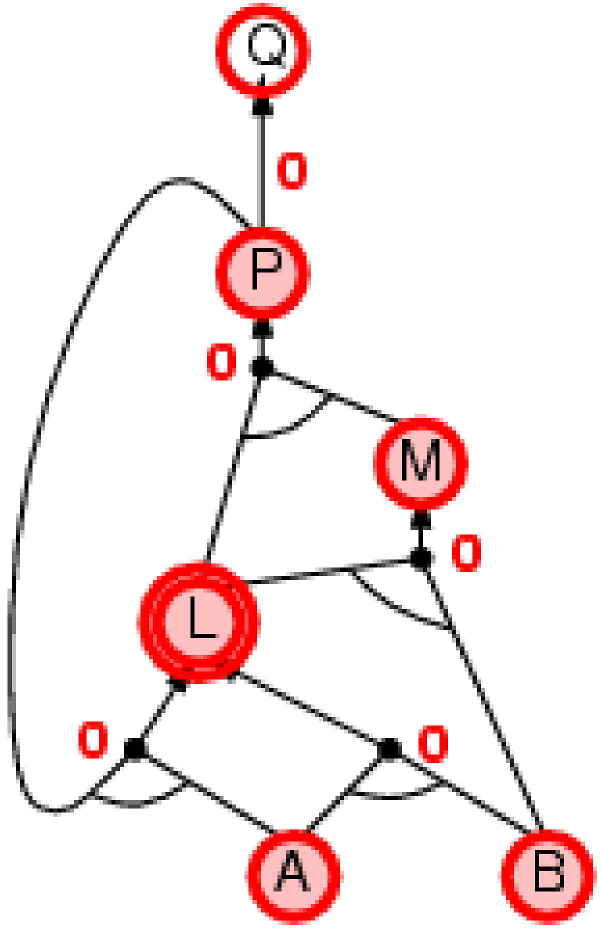
Forward chaining example



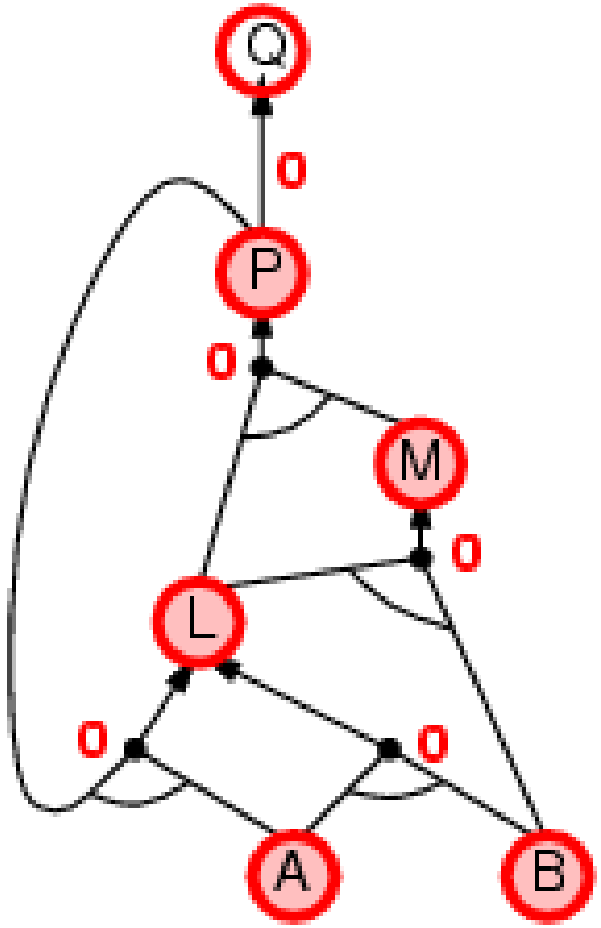
Forward chaining example



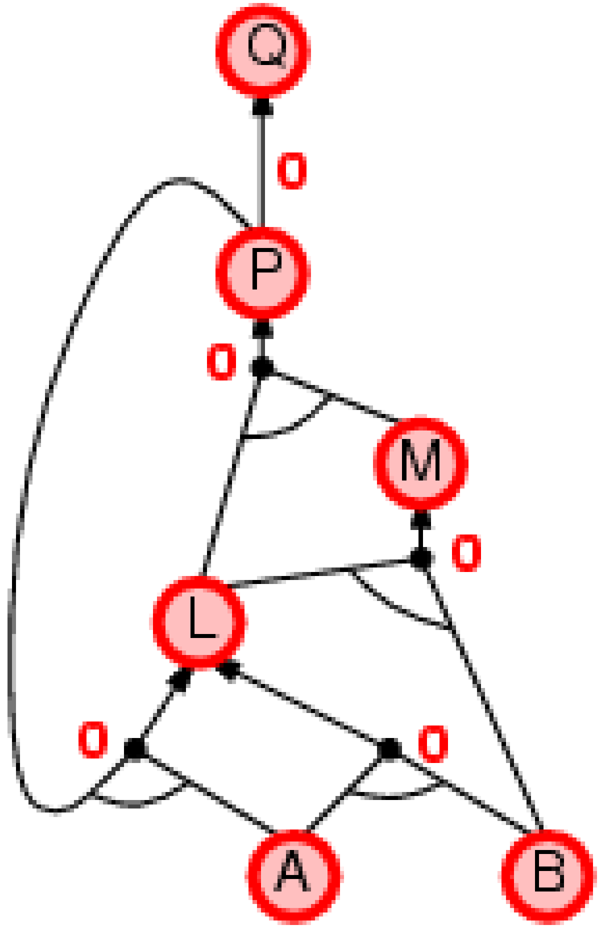
Forward chaining example



Forward chaining example



Forward chaining example



Forward chaining algorithm

```
function PL-FC-ENTAILS?(KB, q) returns true or false
  local variables: count, a table, indexed by clause, initially the number of premises
                   inferred, a table, indexed by symbol, each entry initially false
                   agenda, a list of symbols, initially the symbols known to be true

   while agenda is not empty do
       p ← POP(agenda)
       unless inferred[p] do
           inferred[p] ← true
           for each Horn clause c in whose premise p appears do
               decrement count[c]
               if count[c] = 0 then do
                   if HEAD[c] = q then return true
                   PUSH(HEAD[c], agenda)
   return false
```

- Forward chaining is sound and complete for Horn KB

Proof of completeness

FC derives every atomic sentence that is entailed by *KB*

1. FC reaches a fixed point where no new atomic sentences are derived
2. Consider the final state as a model *m*, assigning true/false to symbols
3. Every clause in the original *KB* is true in *m*

   (if $a_1 \land \ldots \land a_k \Rightarrow b$ is *false* in *m* FC has not reached fixed point)
4. Hence *m* is a model of *KB*
5. If $KB \models q$, *q* is true in every model of *KB*, including *m*

   Perciò è impossibile che FC **non** derivi q; se così fosse, allora per costruzione di **m** esisterebbe un modello (m) di KB in cui *q* = *false* e, per definizione di $\models$, **m** dovrebbe essere anche modello di *q*, che è impossibile

Backward Chaining (BC)

Idea: work backwards from the query q:

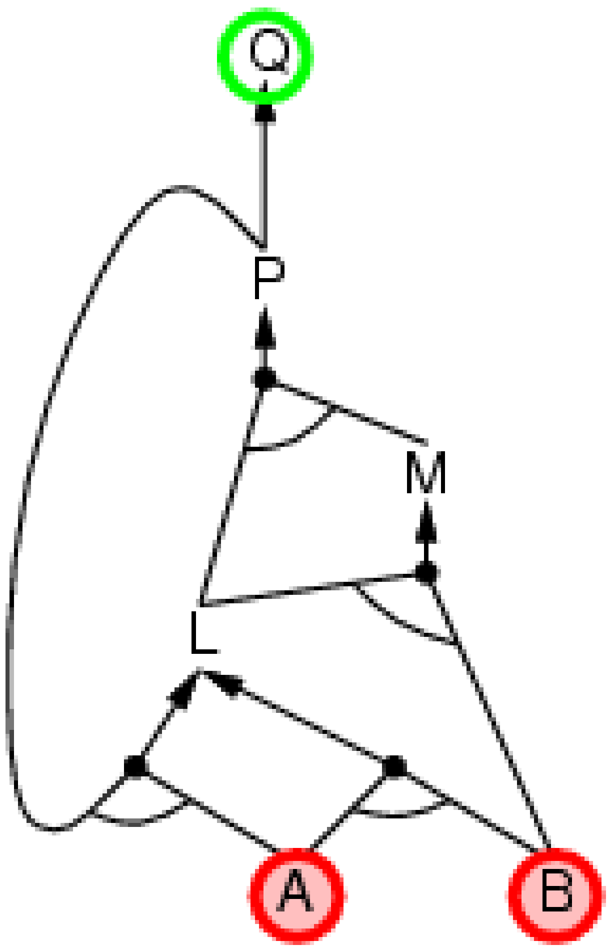
- to prove q by BC,
  - check if q is known already, or
  - prove by BC all premises of some rule concluding q

Avoid loops: check if new subgoal is already on the goal stack

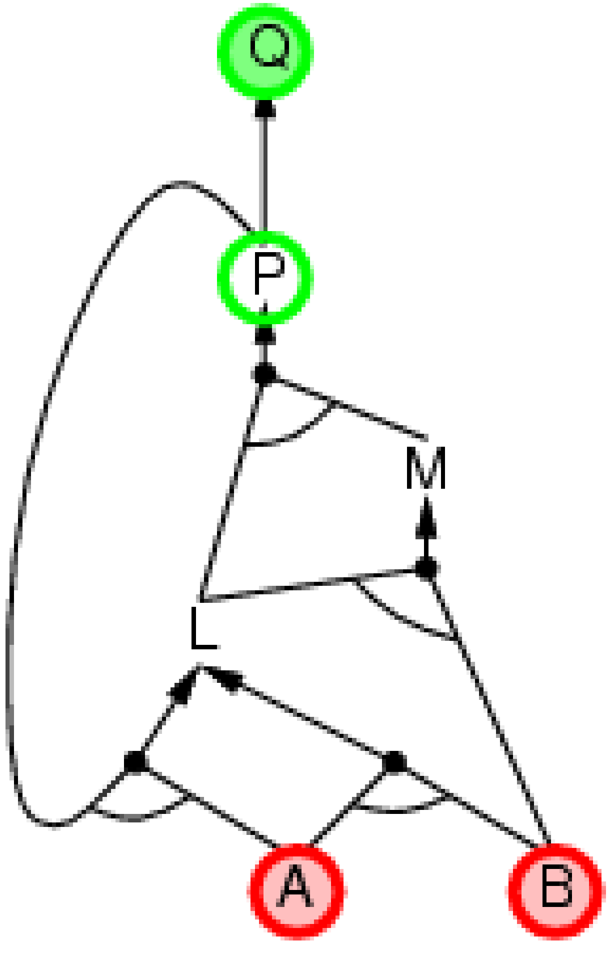
Avoid repeated work: check if new subgoal

1. has already been proved true, or
2. has already failed

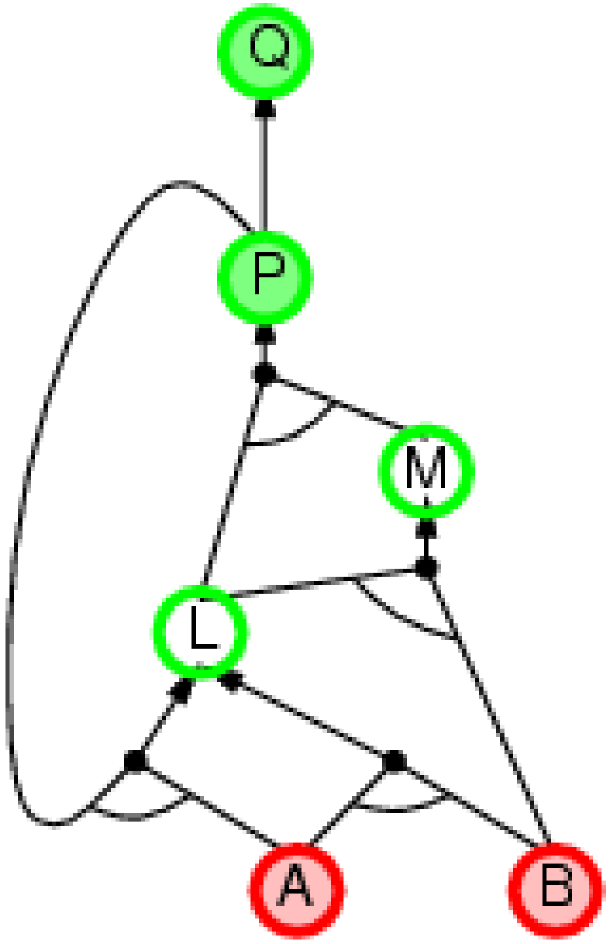
Backward chaining example



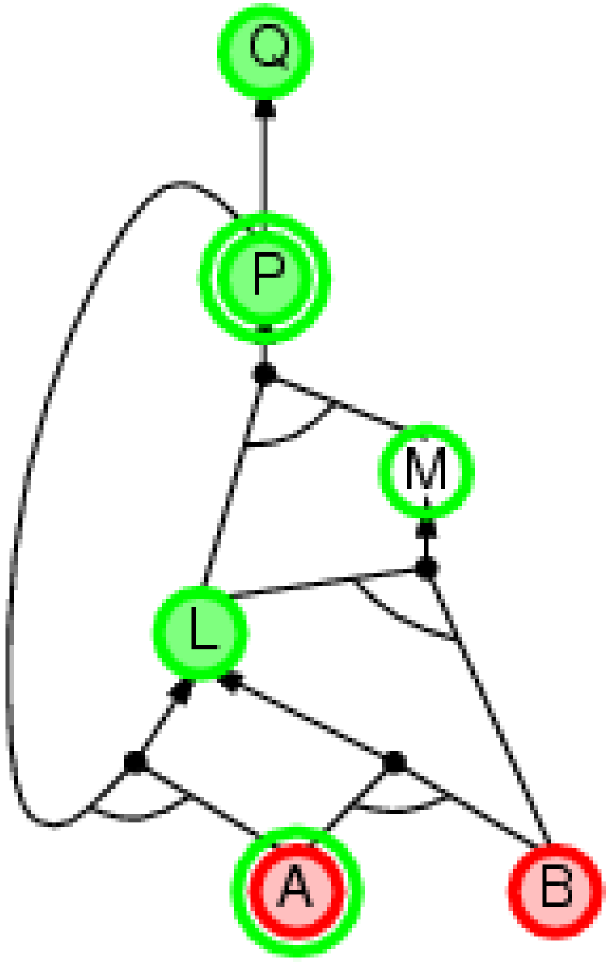
Backward chaining example



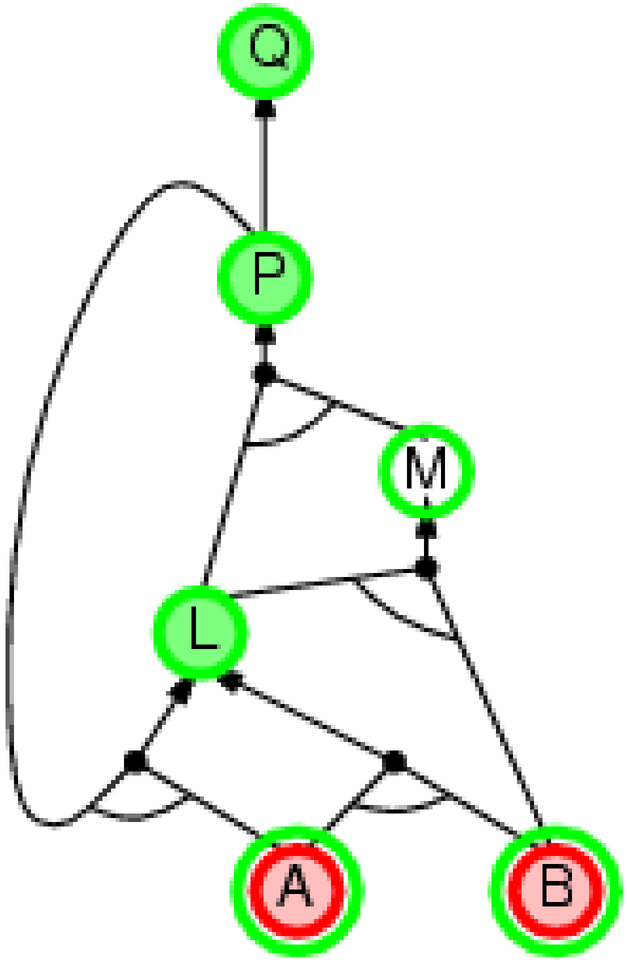
Backward chaining example



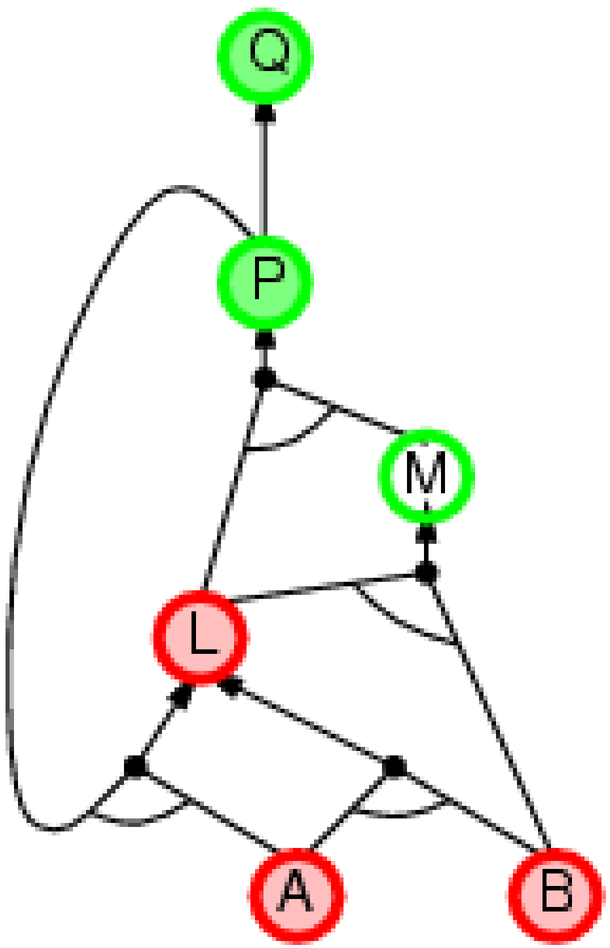
Backward chaining example



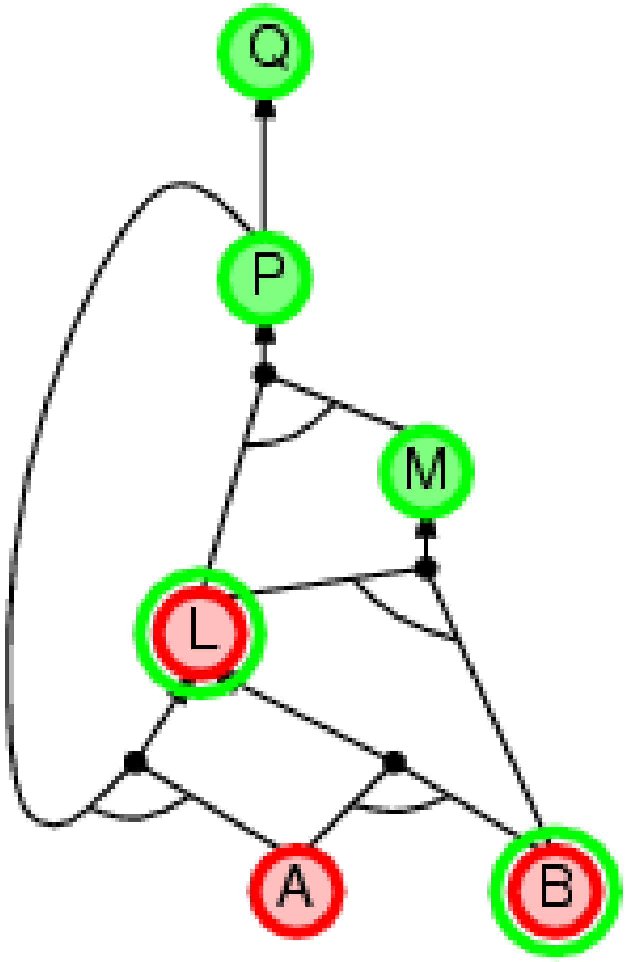
Backward chaining example



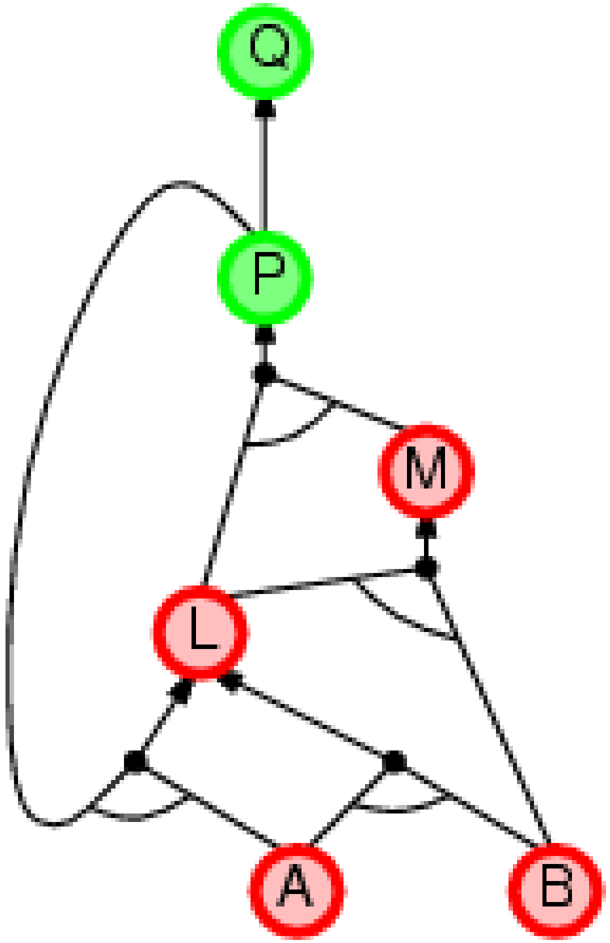
Backward chaining example



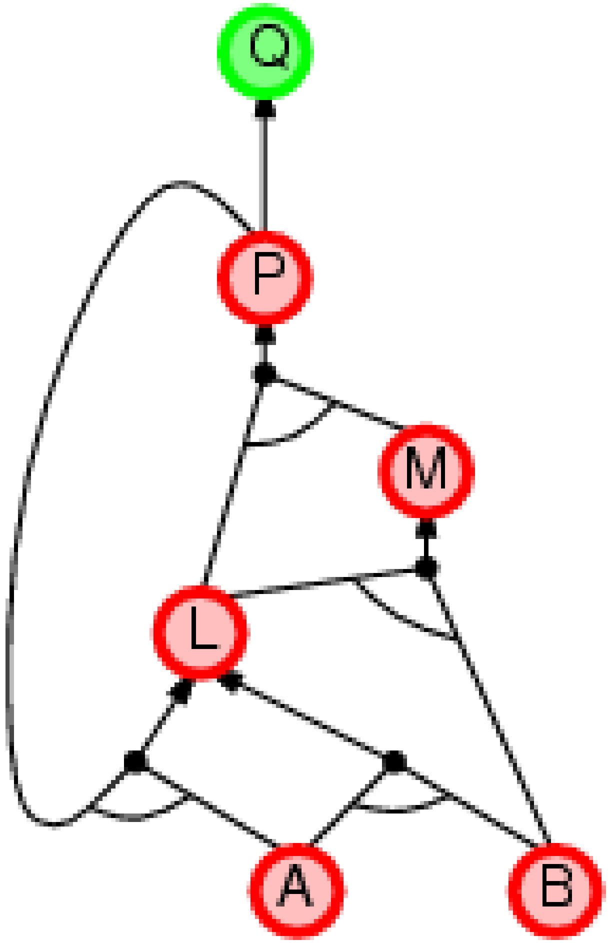
Backward chaining example



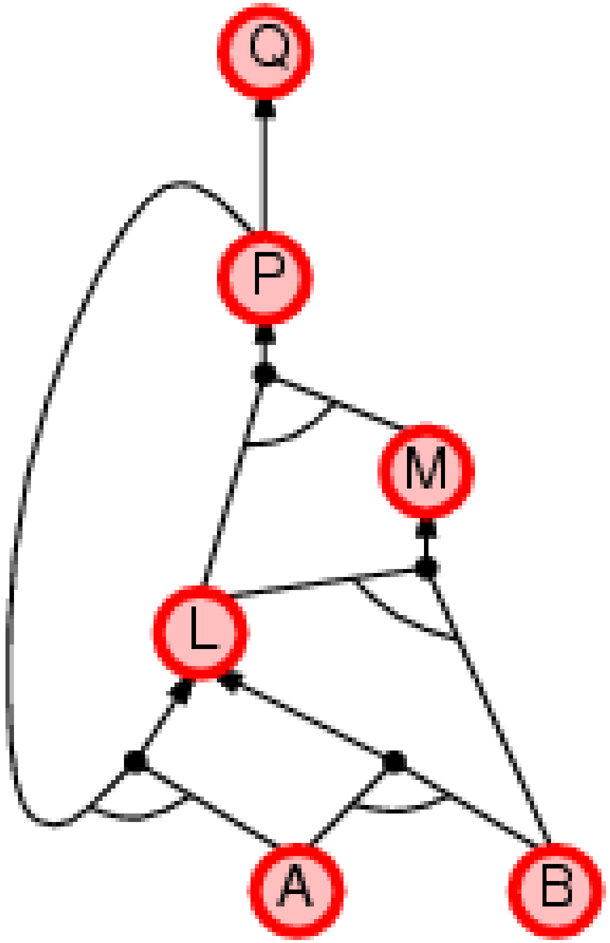
Backward chaining example



Backward chaining example



Backward chaining example



Forward vs. backward chaining

- FC is data-driven, automatic, unconscious processing,
  - e.g., object recognition, routine decisions
- May do lots of work that is irrelevant to the goal
- BC is goal-driven, appropriate for problem-solving,
  - e.g., Where are my keys?
- Complexity of BC can be much less than linear in size of KB

Efficient semantic propositional inference

Two families of efficient algorithms for propositional inference (they work at *semantic* level):

Complete backtracking search algorithms

- DPLL algorithm (Davis, Putnam, Logemann, Loveland)

Incomplete local search algorithms

- `WalkSAT` algorithm

The DPLL algorithm

Determine if an input propositional logic sentence (in CNF) is satisfiable.

Improvements over truth table enumeration:

1. **Early termination**
   A clause is true if any literal is true.
   A sentence (conjunction of clauses) is false if any clause is false.

2. **Pure symbol** heuristic
   Pure symbol: always appears with the same "sign" in all clauses.
   e.g., In the three clauses $(A \vee \neg B)$, $(\neg B \vee \neg C)$, $(C \vee A)$, A and B are pure, C is impure.
   Make a pure symbol literal true.

3. **Unit clause heuristic**
   Unit clause: only one literal in the clause
   The only literal in a unit clause must be true.

The DPLL algorithm

function DPLL-SATISFIABLE?(s) **returns** *true* or *false*
  inputs: s, a sentence in propositional logic

  clauses ← the set of clauses in the CNF representation of s
  symbols ← a list of the proposition symbols in s
  return DPLL(*clauses*, *symbols*, [ ])

---

function DPLL(*clauses*, *symbols*, *model*) **returns** *true* or *false*

  if every clause in *clauses* is true in *model* **then return** *true*
  if some clause in *clauses* is false in *model* **then return** *false*
  P, *value* ← FIND-PURE-SYMBOL(*symbols*, *clauses*, *model*)
  if *P* is non-null **then return** DPLL(*clauses*, *symbols*–*P*, [*P* = *value*|*model*])
  P, *value* ← FIND-UNIT-CLAUSE(*clauses*, *model*)
  if *P* is non-null **then return** DPLL(*clauses*, *symbols*–*P*, [*P* = *value*|*model*])
  P ← FIRST(*symbols*); *rest* ← REST(*symbols*)
  return DPLL(*clauses*, *rest*, [*P* = *true*|*model*]) **or**
      DPLL(*clauses*, *rest*, [*P* = *false*|*model*])

Esercizio DPLL-satisfiable?

(svolto in aula a lezione)

clauses = symbols = …. model = ….

$\neg P \lor \neg Q$
$\neg L \lor \neg D$
$D \lor P$
L
$\neg L \lor F \lor R$
$\neg F \lor \neg R$

Q è puro ….

`WalkSAT` algorithm

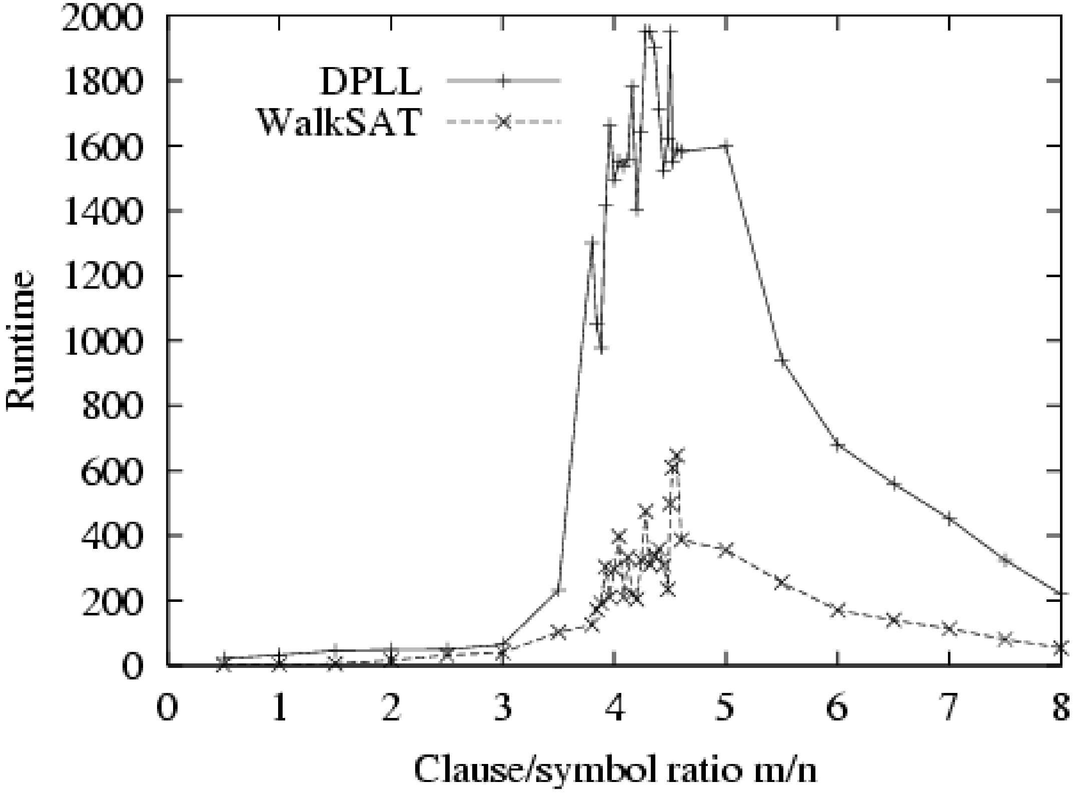
- Incomplete, local search algorithm
- Evaluation function: The min-conflict heuristic of minimizing the number of unsatisfied clauses
- Balance between greediness and randomness

The `WalkSAT` algorithm

```
function WALKSAT(clauses, p, max-flips) returns a satisfying model or failure
   inputs: clauses, a set of clauses in propositional logic
           p, the probability of choosing to do a "random walk" move
           max-flips, number of flips allowed before giving up

   model ← a random assignment of true/false to the symbols in clauses
   for i = 1 to max-flips do
      if model satisfies clauses then return model
      clause ← a randomly selected clause from clauses that is false in model
      with probability p flip the value in model of a randomly selected symbol
            from clause
      else flip whichever symbol in clause maximizes the number of satisfied clauses
   return failure
```

Hard satisfiability problems



- Median runtime for 100 satisfiable random 3-CNF sentences, $n = 50$

Hard satisfiability problems

- Consider random 3-CNF sentences. e.g.,

  $(\neg D \lor \neg B \lor C) \land (B \lor \neg A \lor \neg C) \land (\neg C \lor \neg B \lor E) \land (E \lor \neg D \lor B) \land (B \lor E \lor \neg C)$

  m = number of clauses

  n = number of symbols

  – Hard problems seem to cluster near $m/n = 4.3$ (critical point)

Hard satisfiability problems