Proof methods

Proof methods divide into (roughly) two kinds:

Model checking

truth table enumeration (sound and complete for propositional)

heuristic search in model space (sound but incomplete)

e.g., the GSAT algorithm (Ex. 6.15)

Application of inference rules

Legitimate (sound) generation of new sentences from old

Proof = a sequence of inference rule applications

Can use inference rules as operators in a standard search alg.

Inference Rules – Part I

- **Modus Ponens** or **Implication-Elimination**: (From an implication and the premise of the implication, you can infer the conclusion.)

$$\frac{\alpha \Rightarrow \beta, \quad \alpha}{\beta}$$

- **And-Elimination**: (From a conjunction, you can infer any of the conjuncts.)

$$\frac{\alpha_1 \wedge \alpha_2 \wedge \ldots \wedge \alpha_n}{\alpha_i}$$

- **And-Introduction**: (From a list of sentences, you can infer their conjunction.)

$$\frac{\alpha_1, \; \alpha_2, \quad \ldots, \quad \alpha_n}{\alpha_1 \wedge \alpha_2 \wedge \ldots \wedge \alpha_n}$$

- **Or-Introduction**: (From a sentence, you can infer its disjunction with anything else at all.)

$$\frac{\alpha_i}{\alpha_1 \vee \alpha_2 \vee \ldots \vee \alpha_n}$$

Inference Rules – Part II

- **Double-Negation Elimination**: (From a doubly negated sentence, you can infer a positive sentence.)

$$\frac{\neg\neg\alpha}{\alpha}$$

- **Unit Resolution:** (From a disjunction, if one of the disjuncts is false, then you can infer the other one is true.)

$$\frac{\alpha \vee \beta, \qquad \neg\beta}{\alpha}$$

- **Resolution**: (This is the most difficult. Because β cannot be both true and false, one of the other disjuncts must be true in one of the premises. Or equivalently, implication is transitive.)

$$\frac{\alpha \vee \beta, \qquad \neg\beta \vee \gamma}{\alpha \vee \gamma} \quad \text{or equivalently} \quad \frac{\neg\alpha \Rightarrow \beta, \qquad \beta \Rightarrow \gamma}{\neg\alpha \Rightarrow \gamma}$$

Wumpus world: example

- **Facts:** Percepts inject (TELL) facts into the KB
  - [stench at 1,1 and 2,1] → S1,1 ; S2,1
- **Rules:** if square has no stench then neither the square or adjacent square contain the wumpus
  - R1: $!S1,1 \Rightarrow !W1,1 \wedge !W1,2 \wedge !W2,1$
  - R2: $!S2,1 \Rightarrow !W1,1 \wedge !W2,1 \wedge !W2,2 \wedge !W3,1$
  - …
- **Inference:**
  - KB contains !S1,1 then using Modus Ponens we infer

    $!W1,1 \wedge !W1,2 \wedge !W2,1$
  - Using And-Elimination we get: !W1,1 !W1,2 !W2,1
  - …

Resolution

Conjunctive Normal Form (CNF)

conjunction of disjunctions of literals clauses

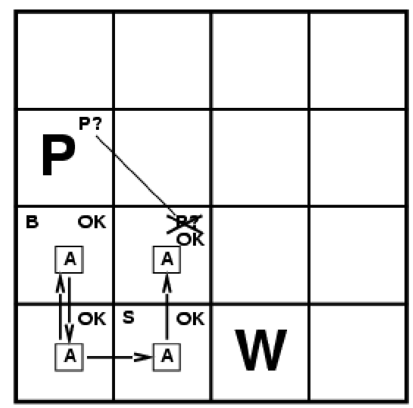
E.g., $(A \vee \neg B) \wedge (B \vee \neg C \vee \neg D)$

- Resolution inference rule (for CNF):

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

where l_i and m_j are complementary literals.

E.g., $$\frac{P_{1,3} \vee P_{2,2}, \qquad \neg P_{2,2}}{P_{1,3}}$$



- Resolution is sound and complete for propositional logic

Conversion to Conjunctive Normal Form

1. Eliminate $\Leftrightarrow$, replacing $\alpha \Leftrightarrow \beta$ with $(\alpha \Rightarrow \beta) \wedge (\beta \Rightarrow \alpha)$.

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

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

4. Apply distributivity law ($\wedge$ over $\vee$) and flatten:

CNF is required to apply RESOLUTION

Conversion to Conjunctive Normal Form - Example

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

1. Eliminate $\Leftrightarrow$, replacing $\alpha \Leftrightarrow \beta$ with $(\alpha \Rightarrow \beta) \wedge (\beta \Rightarrow \alpha)$.

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

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

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

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

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

4. Apply distributivity law ($\wedge$ over $\vee$) and flatten:

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

Manual Application of resolution

- In order to show KB entails **α** the steps are as follows:
  - Convert the sentences of KB into CNF
  - Convert **α** into CNF
  - Prove α by using proof **by contradiction**,
  - Namely, show ***KB*** $\wedge \neg$ **α** is **unsatisfiable**
  - This will mean showing that the resolvent set is empty
  - If the resolvent set is empty, then this shows the contradiction
  - Therefore, **α** must be entailed by KB

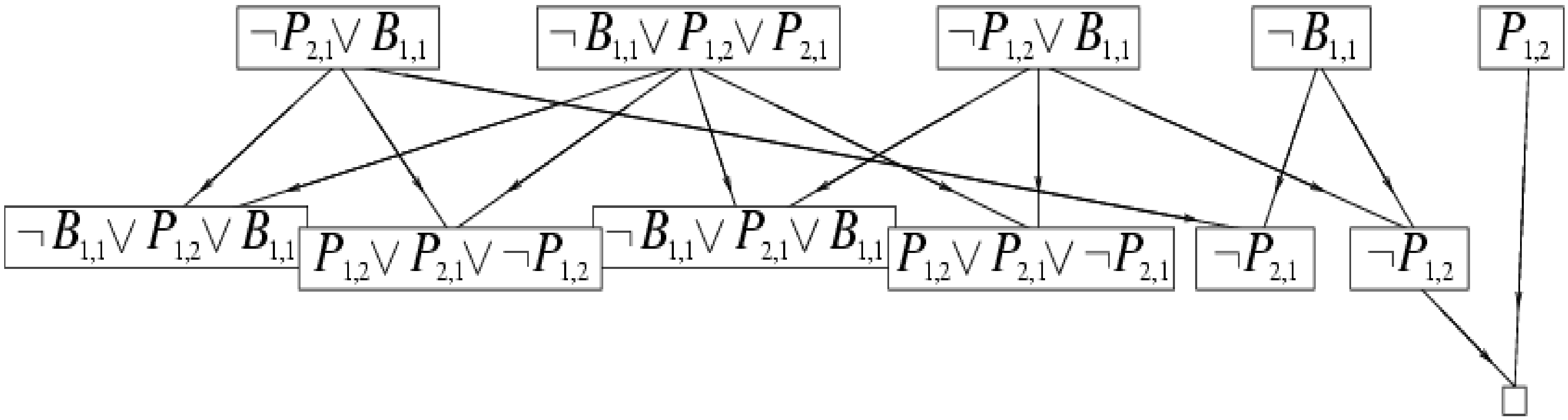
Resolution algorithm

- Mechanical way to prove α by using proof by contradiction, i.e., show ***KB* ∧ ¬ α** is **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 Ci, Cj in clauses do
            resolvents ← PL-RESOLVE(Ci, Cj)
            if resolvents contains the empty clause then return true
            new ← new ∪ resolvents
        if new ⊆ clauses then return false
        clauses ← clauses ∪ new
```

Resolution example

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



KB in Horn Form – Useful Special Case to Analyze

- **Horn Form** (restricted)

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

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

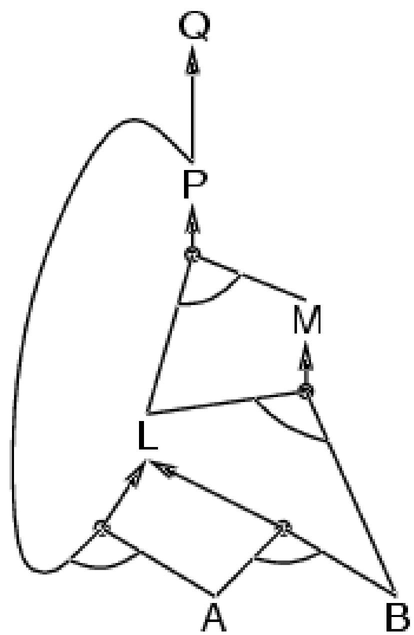
$$\frac{\alpha_1, \ldots, \alpha_n, \qquad\qquad \alpha_1 \wedge \ldots \wedge \alpha_n \Rightarrow \beta}{\beta}$$

- Can be used with **forward chaining** or **backward chaining**.
- These algorithms are very natural and run in **linear** time
- Far more efficient than RESOLUTION – which is more general, but more expensive
- PROLOG programming language uses Horn Clauses

Forward chaining

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

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

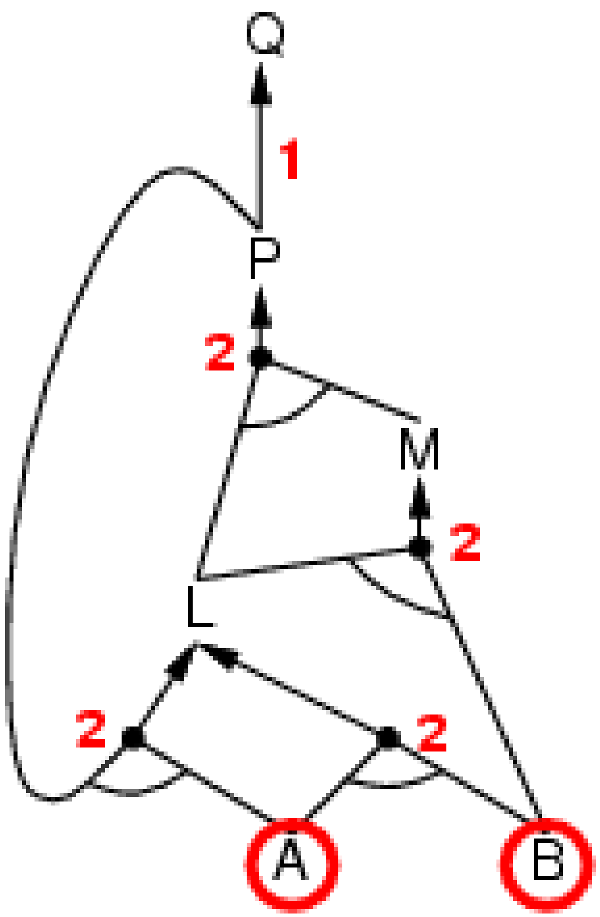


Forward chaining algorithm

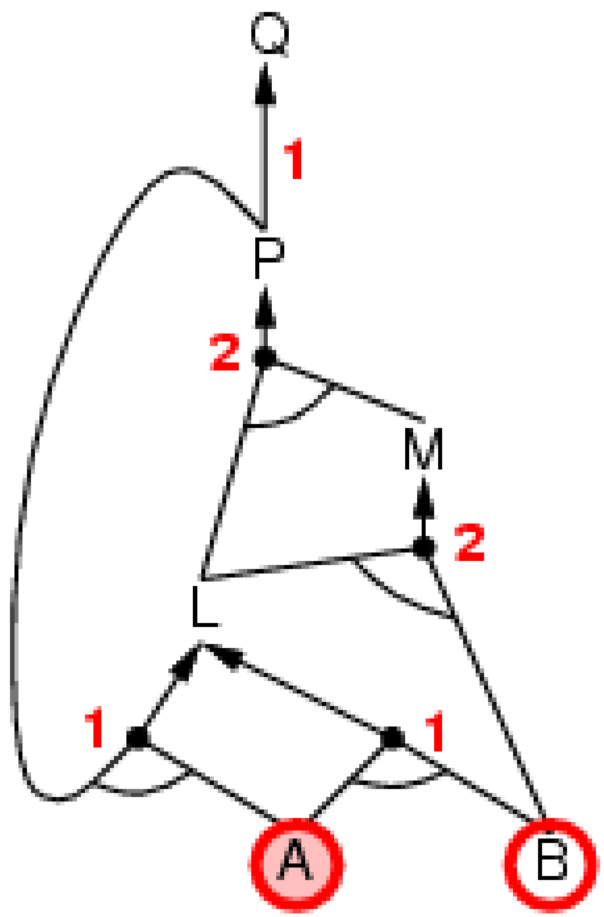
```
function PL-FC-ENTAILS?(KB, q) returns true or false
  local variables: count, a table, indexed by clause, initially the number of premises
                   inferred, a table, indexed by symbol, each entry initially false
                   agenda, a list of symbols, initially the symbols known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

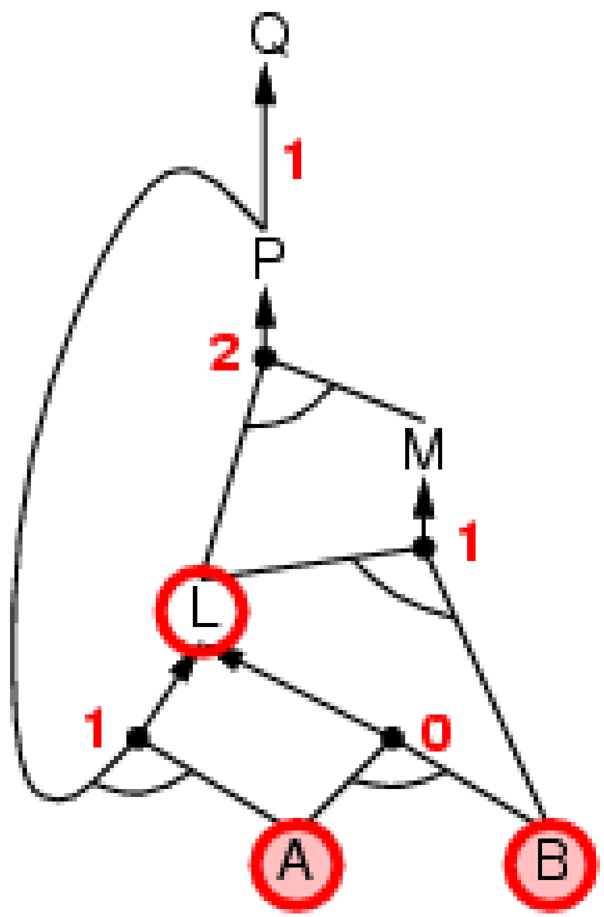
Forward chaining example



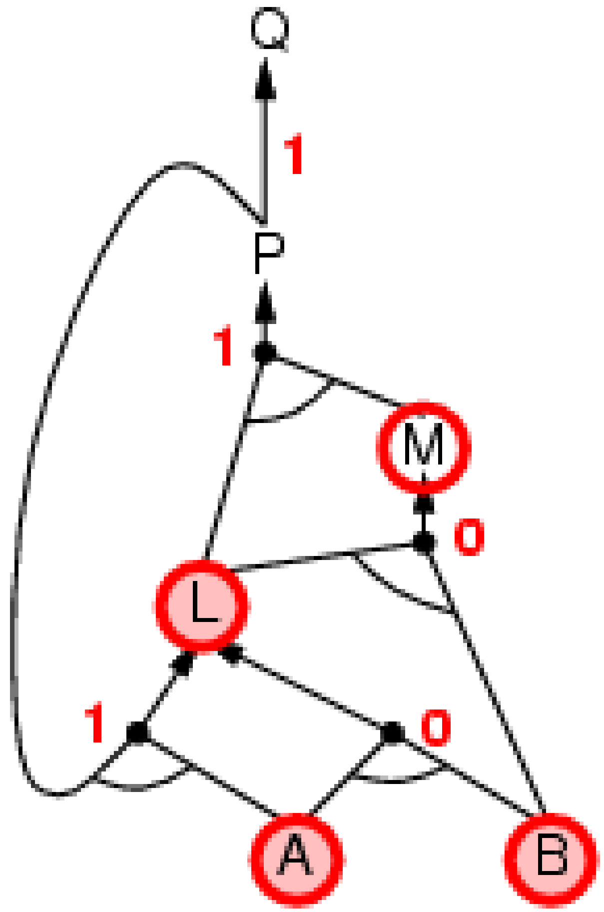
Forward chaining example



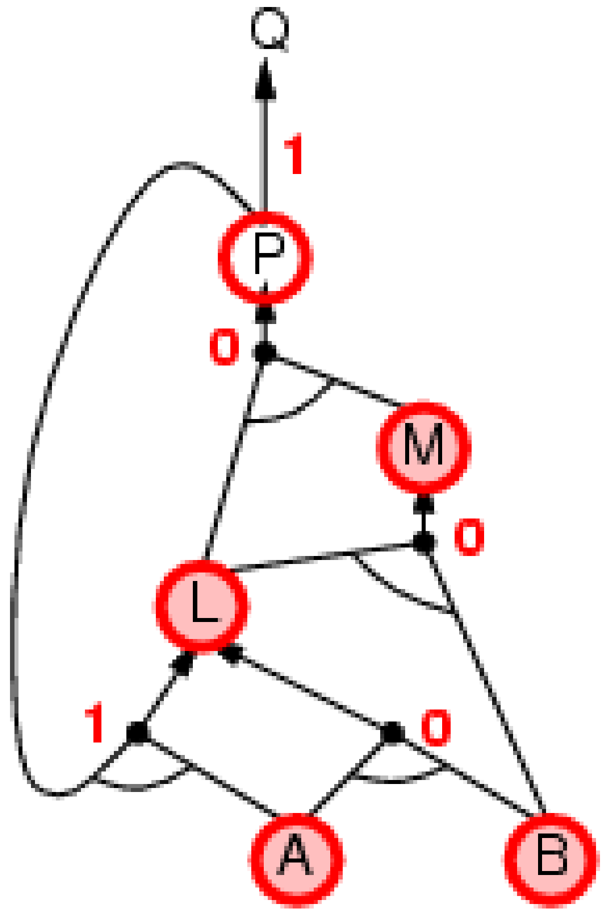
Forward chaining example



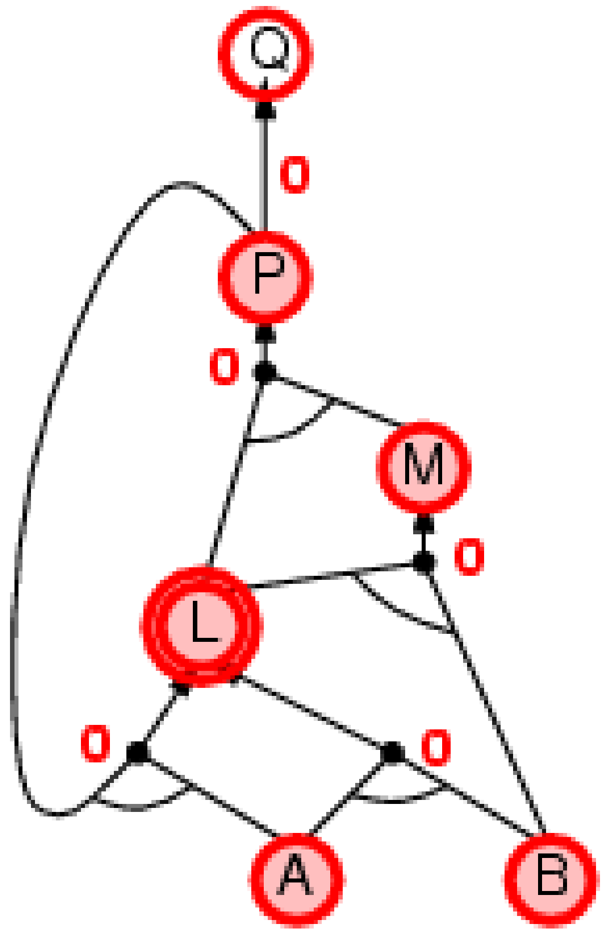
Forward chaining example



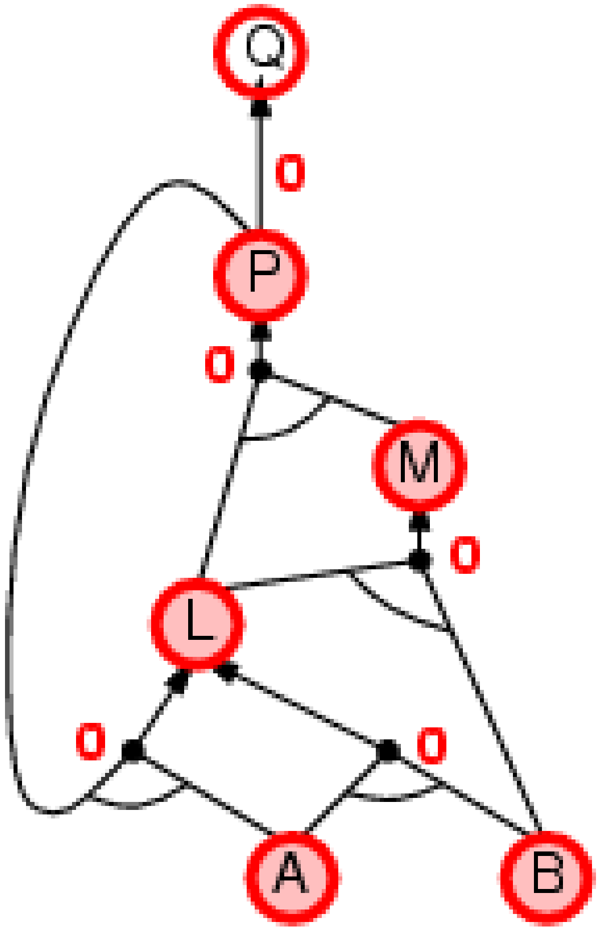
Forward chaining example



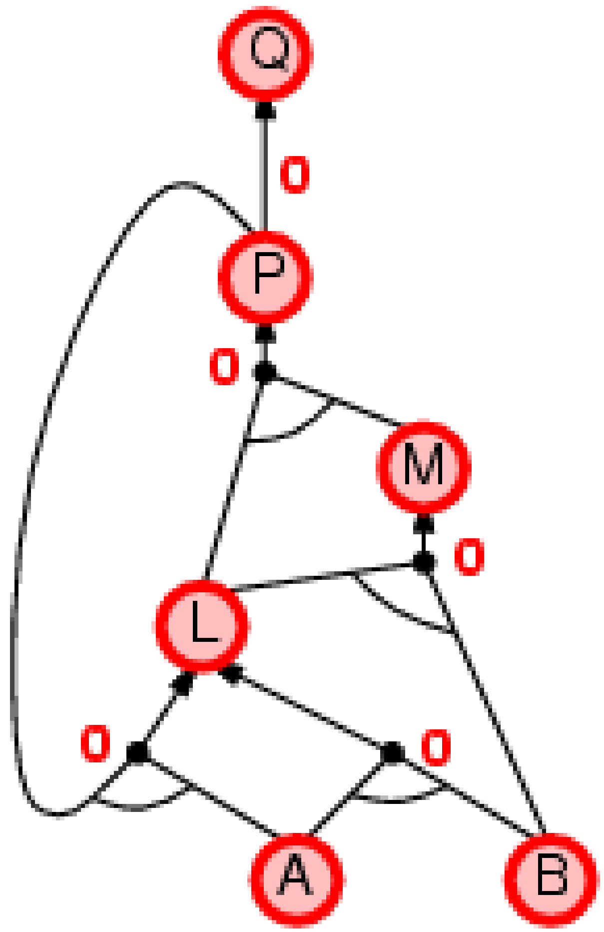
Forward chaining example



Forward chaining example



Forward chaining example



Proof of completeness

- FC derives every atomic sentence that is entailed by *KB*
  1. FC eventually 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*

     $a_1 \wedge \ldots \wedge a_k \Rightarrow b$
  4. Hence *m* is a model of *KB*
  5. If $KB \models q$, *q* is true in every model of *KB*, including *m*

Backward chaining

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*

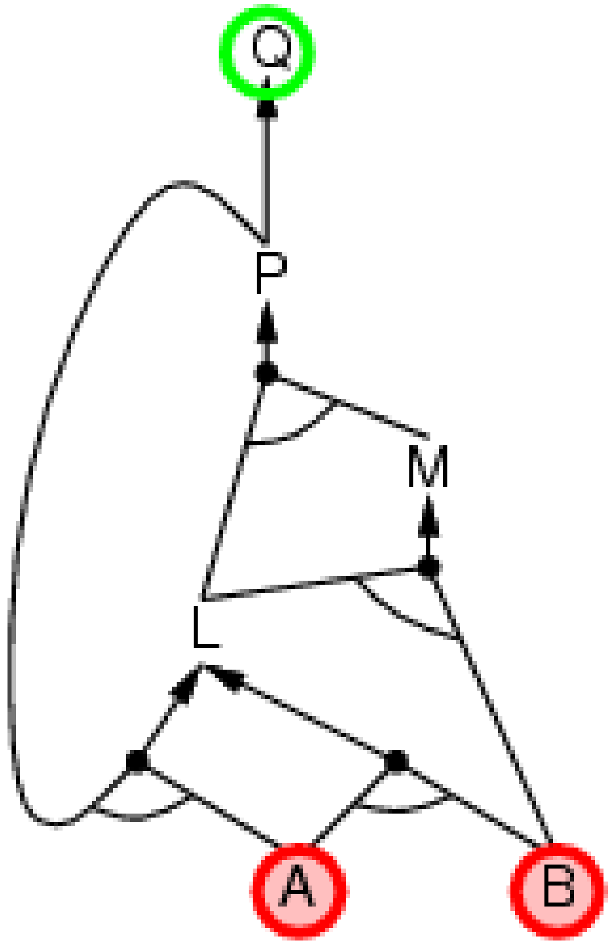
Backward chaining is sound and complete **for Horn KB**

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

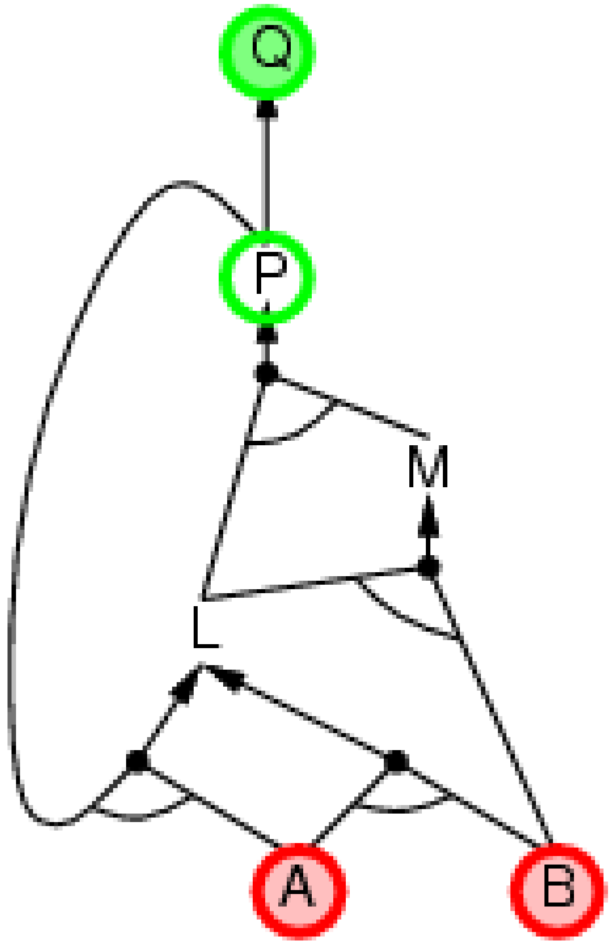
Avoid repeated work: check if new subgoal

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

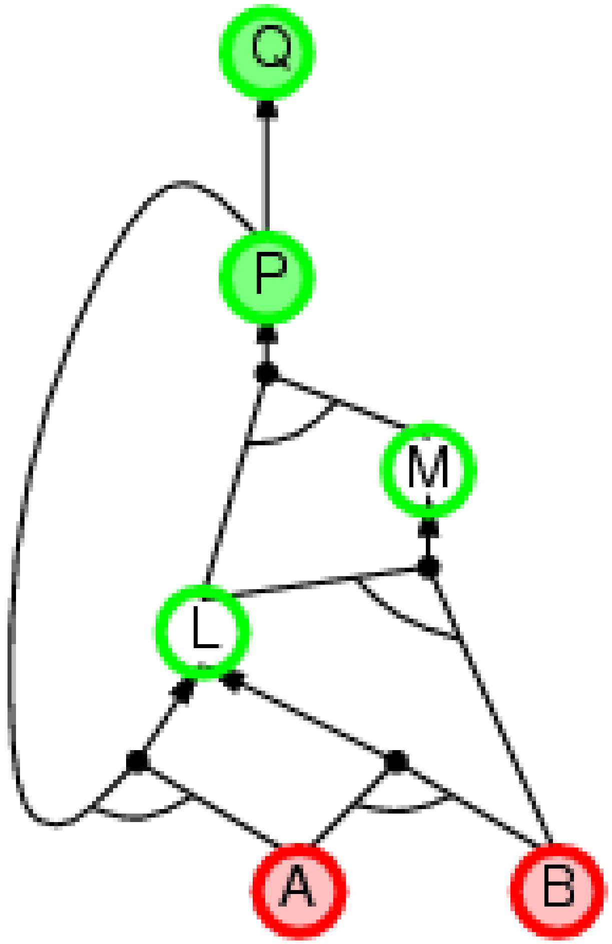
Backward chaining example



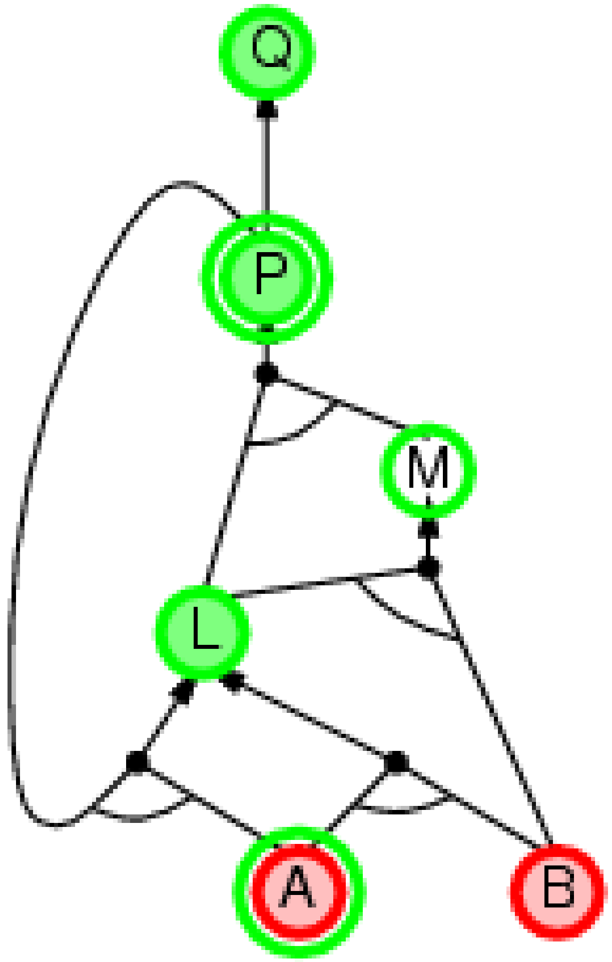
Backward chaining example



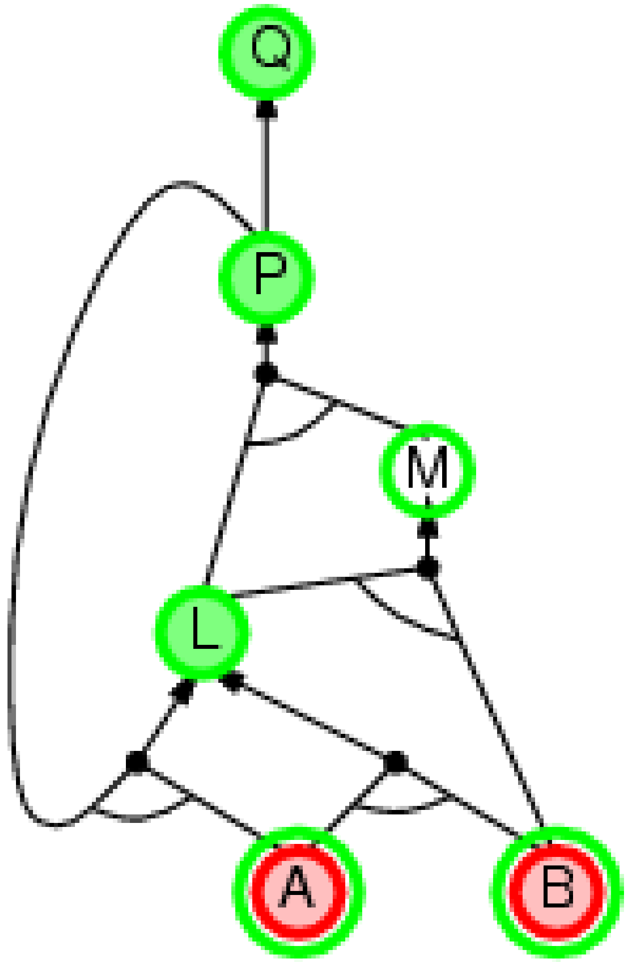
Backward chaining example



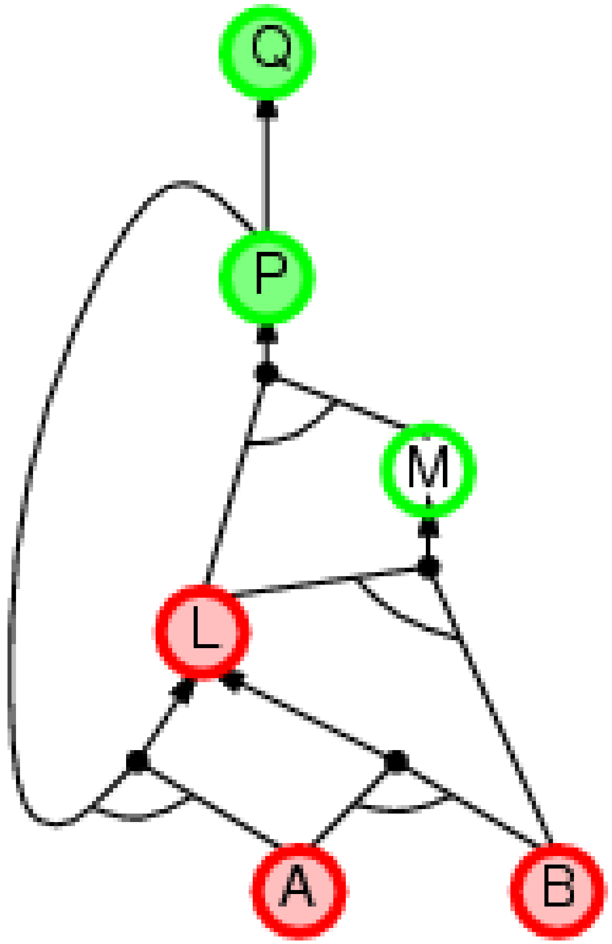
Backward chaining example



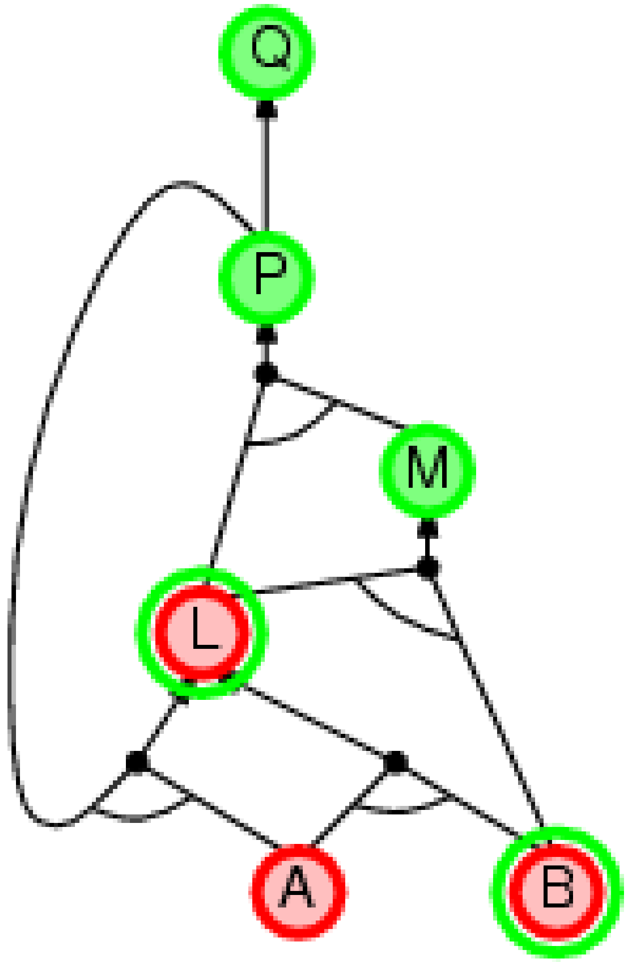
Backward chaining example



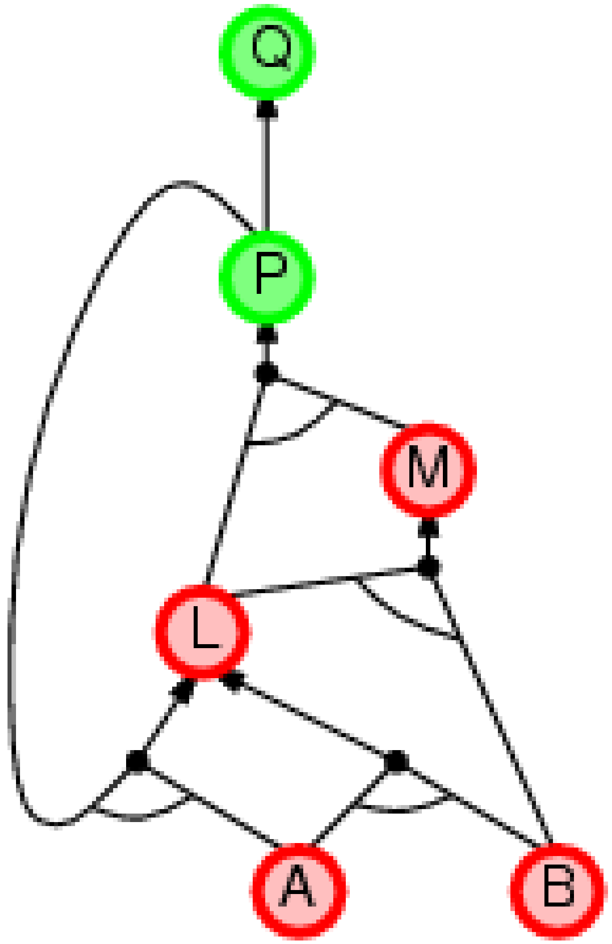
Backward chaining example



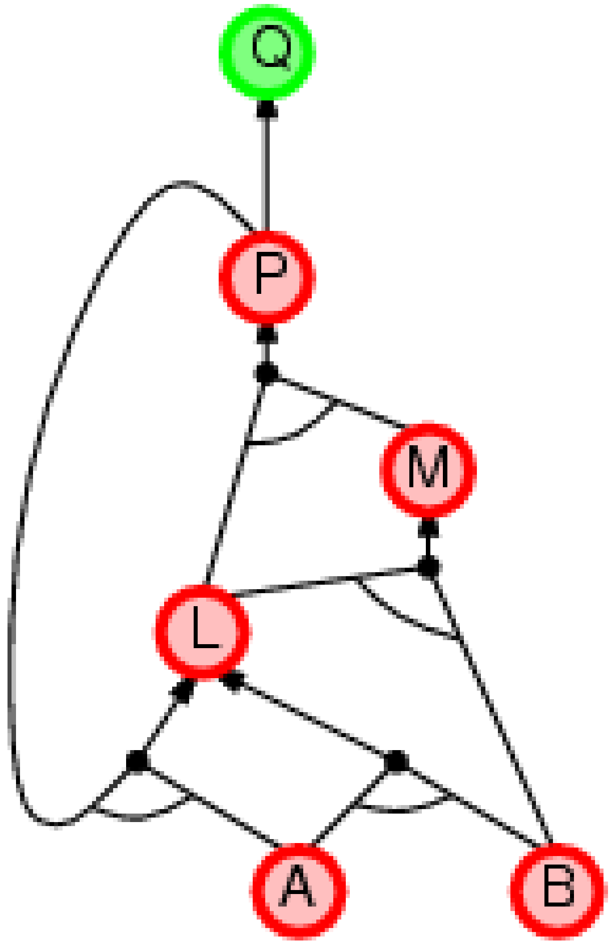
Backward chaining example



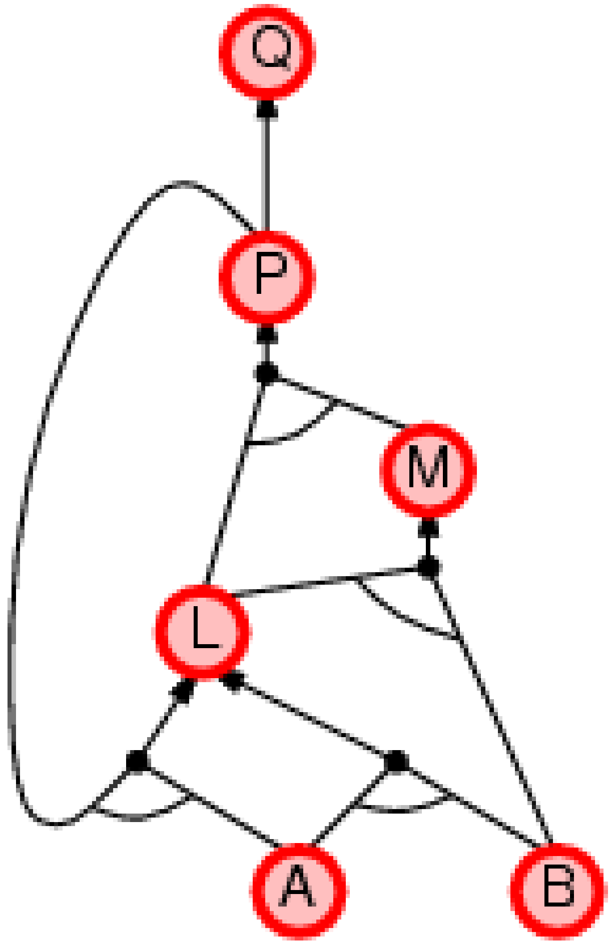
Backward chaining example



Backward chaining example



Backward chaining example



Forward vs. backward chaining

- FC is **data-driven**, automatic, unconscious processing,
  - e.g., object recognition, decisions related to formulating responses to changing conditions
  - 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? How do I get into a PhD program?
  - Complexity of BC can be much less than linear in size of KB

Limitations of Propositional Logic

1. It is too weak, i.e., has very limited expressiveness:

- Each rule has to be represented for each situation: e.g., "don't go forward if the wumpus is in front of you" takes 64 rules

2. It cannot keep track of changes:

- If one needs to track changes, e.g., where the agent has been before then we need a timed-version of each rule. To track 100 steps we'll then need 6400 rules for the previous example.

Its **hard to write and maintain** such a huge rule-base

Inference becomes intractable

Summary

Logical agents apply inference to a knowledge base
to derive new information and make decisions

Basic concepts of logic:

- syntax: formal structure of sentences
- semantics: truth of sentences wrt models
- entailment: necessary truth of one sentence given another
- inference: deriving sentences from other sentences
- soundess: derivations produce only entailed sentences
- completeness: derivations can produce all entailed sentences

Wumpus world requires the ability to represent partial and negated information, reason by cases, etc.

Propositional logic suffices for some of these tasks

Truth table method is sound and complete for propositional logic