Adapted from AIMA slides

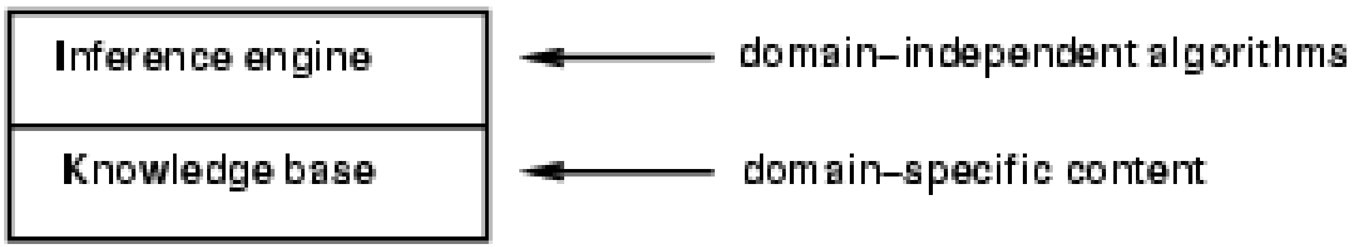
Logic: automated reasoning, provers

Peter Antal
antal@mit.bme.hu

Outline

- Truth and proofs
- Inference rules and theorem proving
  - forward chaining
  - backward chaining
  - Resolution
    - Resolution as elementary inference step
    - Resolution as general inference method
    - Conversion to conjuctive normal form (CNF)
    - Resolution heuristics
- Exercises

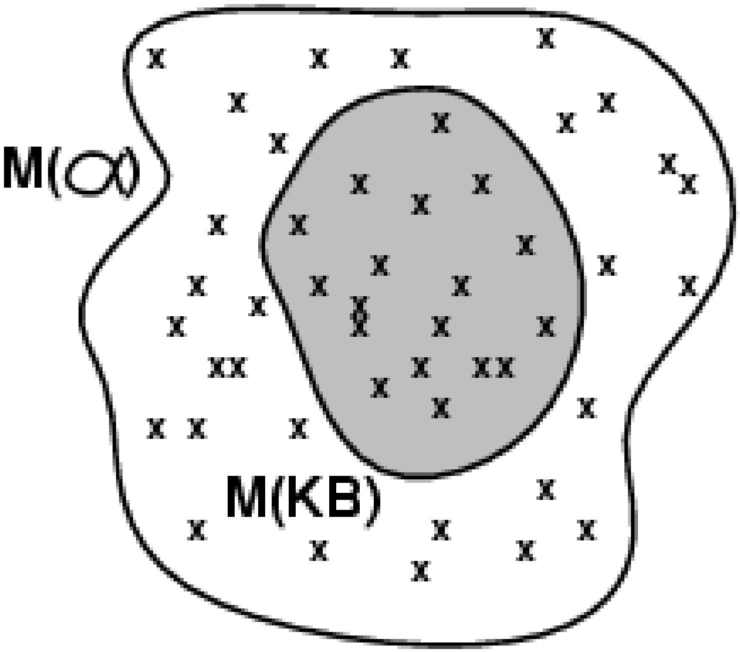
Reminder: Knowledge bases



- Knowledge base = set of sentences in a formal language
- Declarative approach to building an agent (or other system):
  - Tell it what it needs to know
  -
- Then it can Ask itself what to do – answers should follow from the KB
-
- Agents can be viewed at the knowledge level
  i.e., what they know, regardless of how implemented
- Or at the implementation level
  - i.e., data structures in KB and algorithms that manipulate them
  -

Reminder: Models

- Logicians typically think in terms of models, which are formally structured worlds with respect to which truth can be evaluated
-
- We say *m* is a model of a sentence
- $M(\alpha)$ is the set of all models of α
-
- Then KB $\models$ α iff $M(KB) \subseteq M(\alpha)$
-
  - E.g. *KB* = Giants won and Reds won α = Giants won
  -



Reminder: truth vs. proof

- Soundness: *i* is sound if whenever $KB \vdash_i \alpha$, it is also true that $KB \models \alpha$
-
- Completeness: *i* is complete if whenever $KB \models \alpha$, it is also true that $KB \vdash_i \alpha$
-
- Preview: we will define a logic (first-order logic) which is expressive enough to say almost anything of interest, and for which there exists a sound and complete inference procedure.
-
- That is, the procedure will answer any question whose answer follows from what is known by the *KB*.
-

Forward and backward chaining

- Horn Form (restricted)
  KB = conjunction of Horn 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

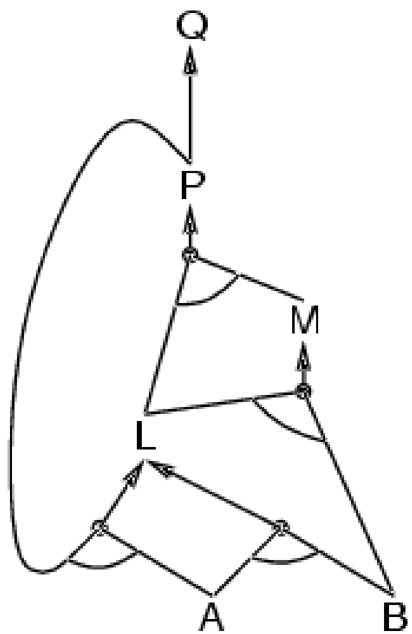
$$\frac{\alpha_1, \ldots, \alpha_n, \qquad \alpha_1 \wedge \ldots \wedge \alpha_n \Rightarrow \beta}{\beta}$$

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

Forward chaining

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

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

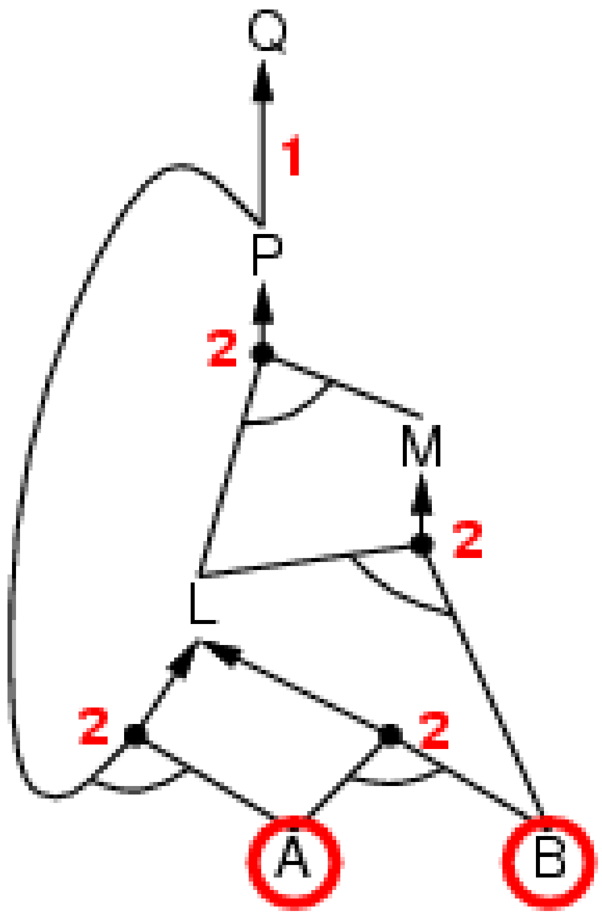


Forward chaining algorithm

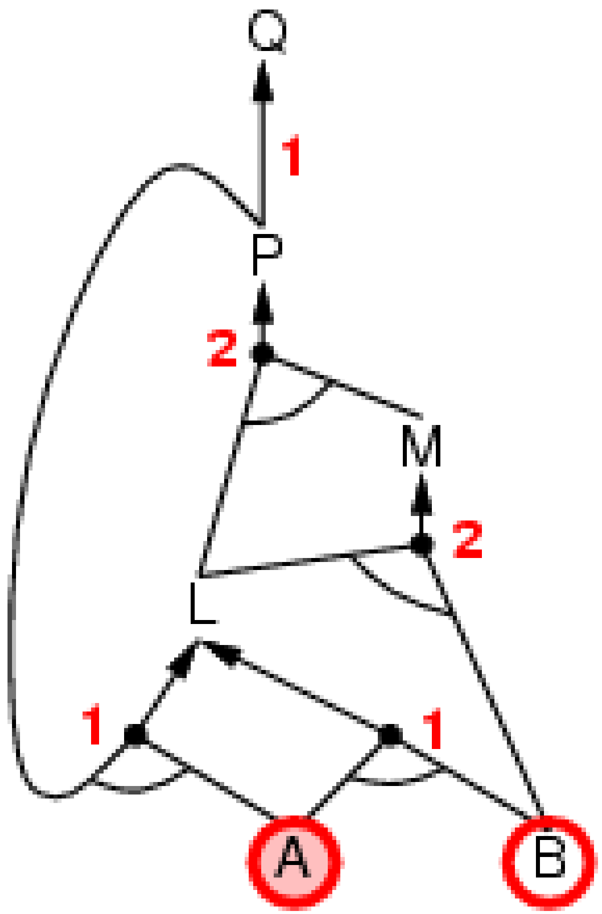
```
function PL-FC-ENTAILS?(KB, q) returns true or false
  local variables: count, a table, indexed by clause, initially the number of premises
                   inferred, a table, indexed by symbol, each entry initially false
                   agenda, a list of symbols, initially the symbols known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

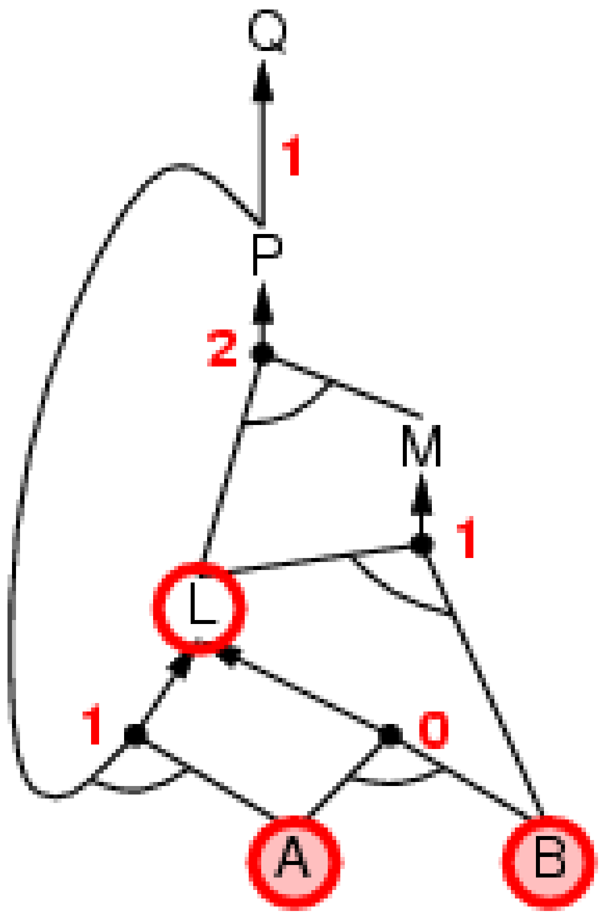
Forward chaining example



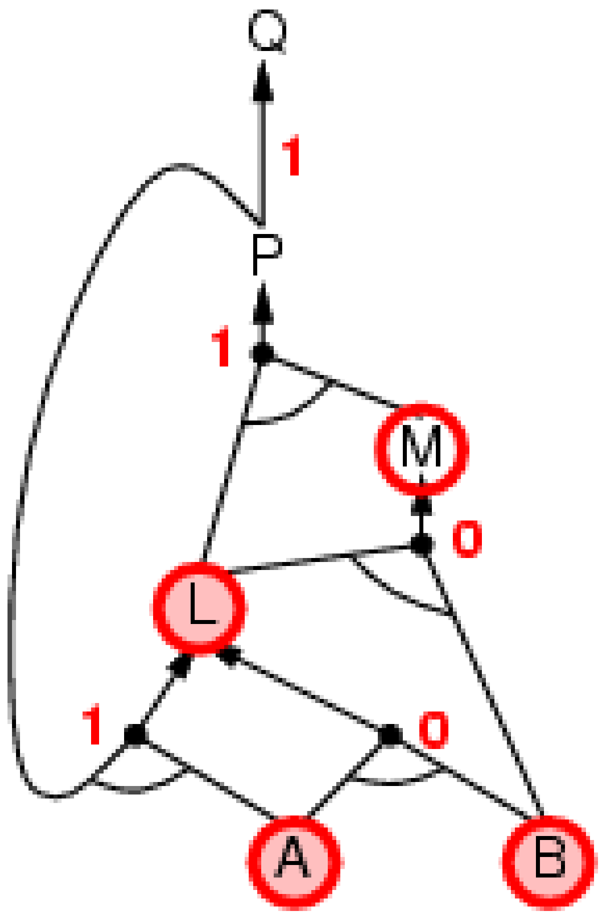
Forward chaining example



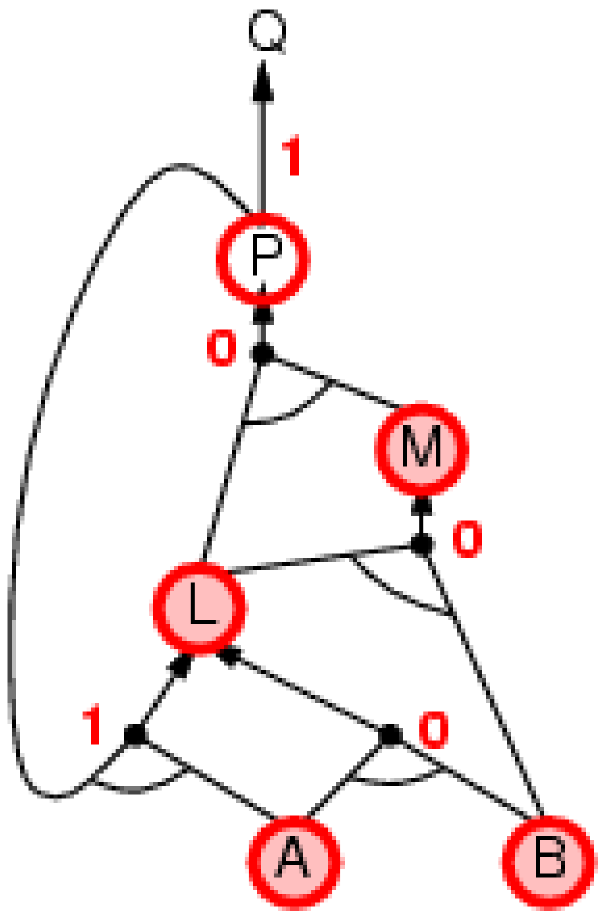
Forward chaining example



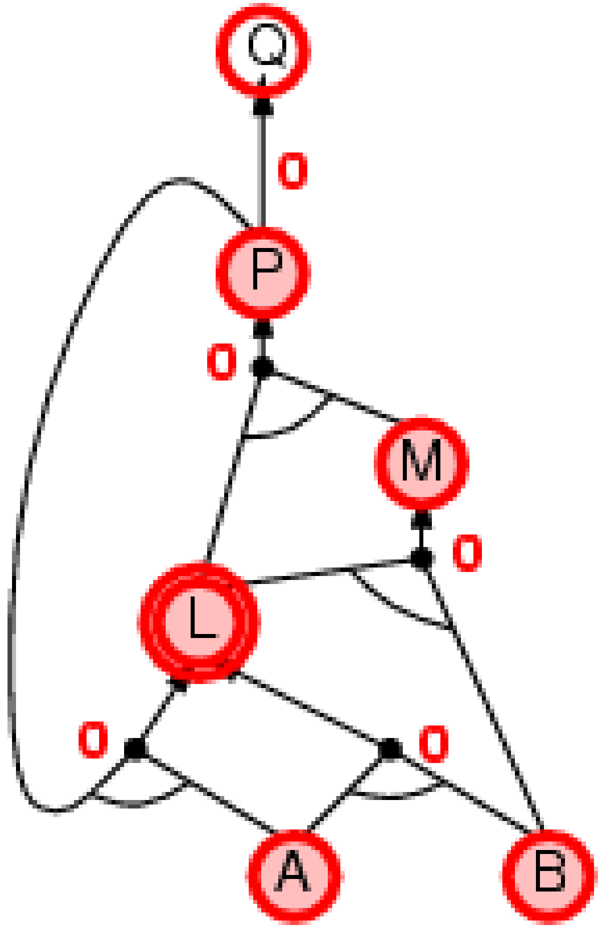
Forward chaining example



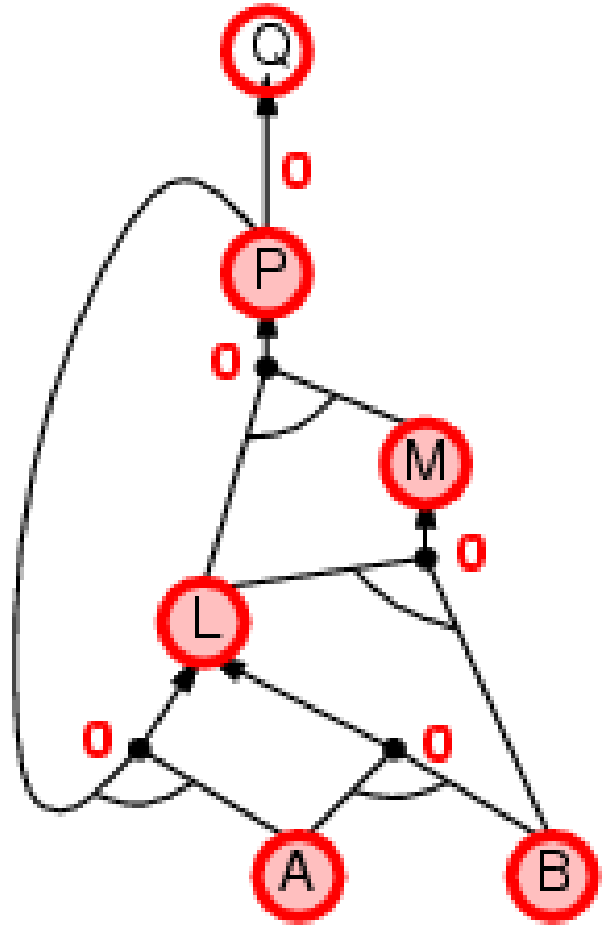
Forward chaining example



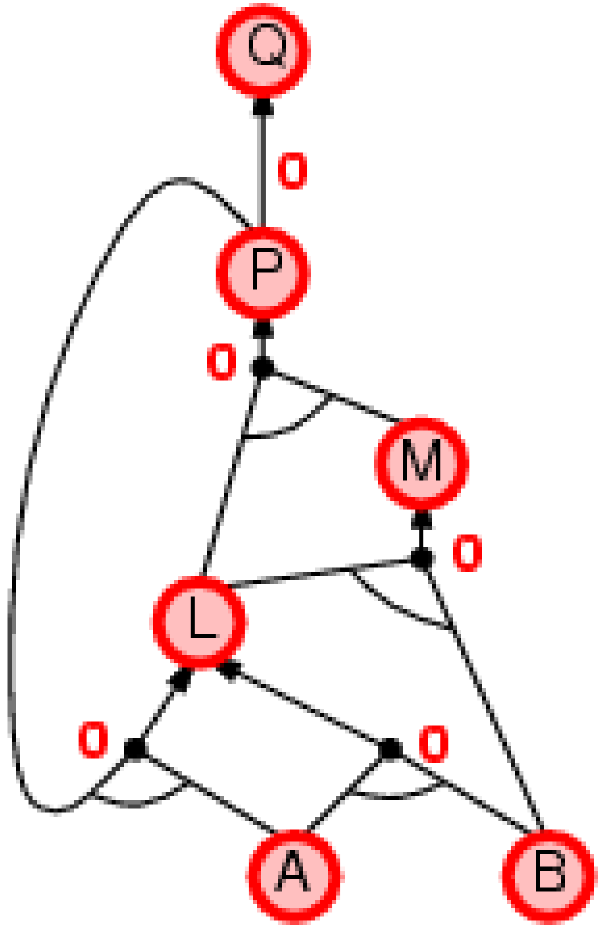
Forward chaining example



Forward chaining example



Forward chaining example



Backward chaining

Idea: work backwards from the query q:

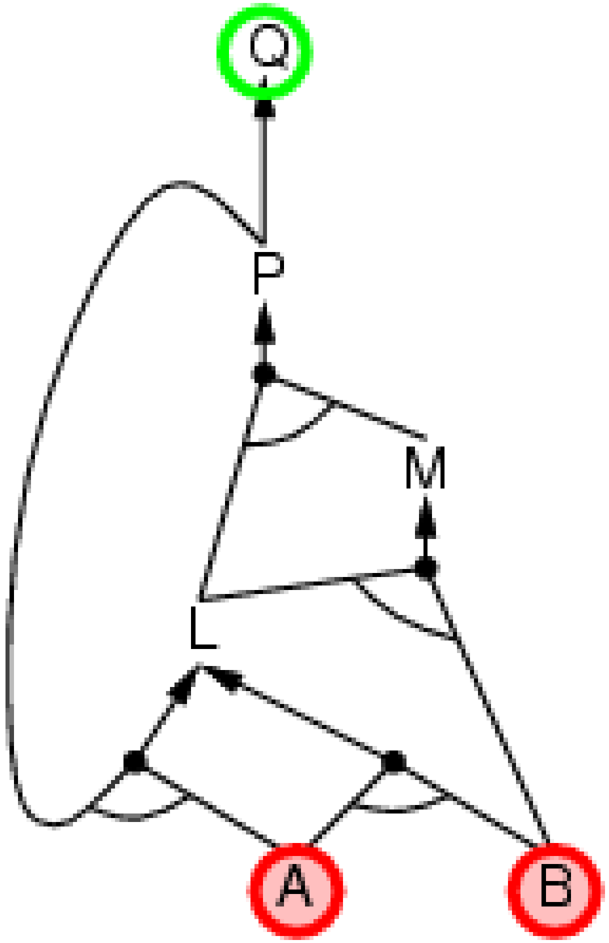
to prove q by BC,
check if q is known already, or
prove by BC all premises of some rule concluding q

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

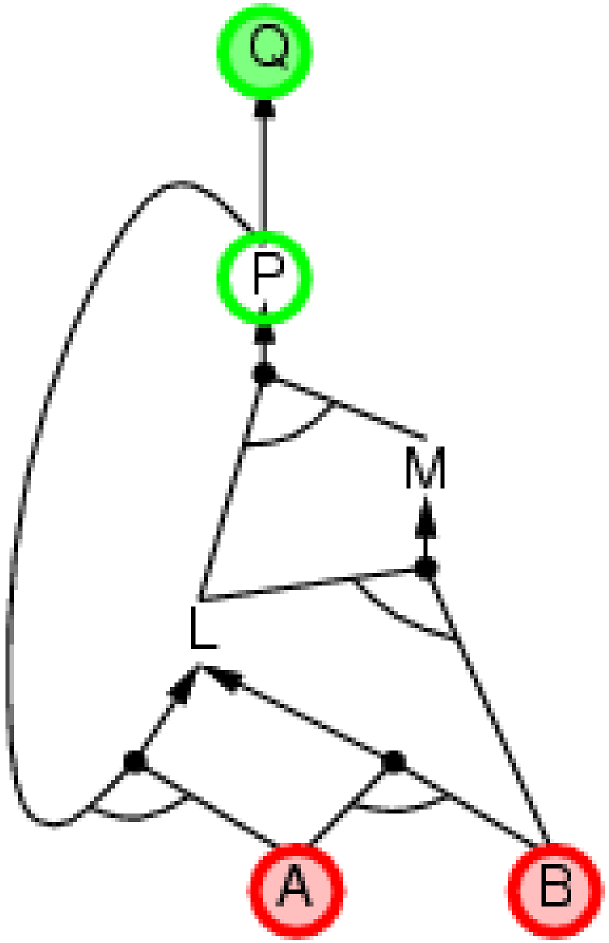
Avoid repeated work: check if new subgoal

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

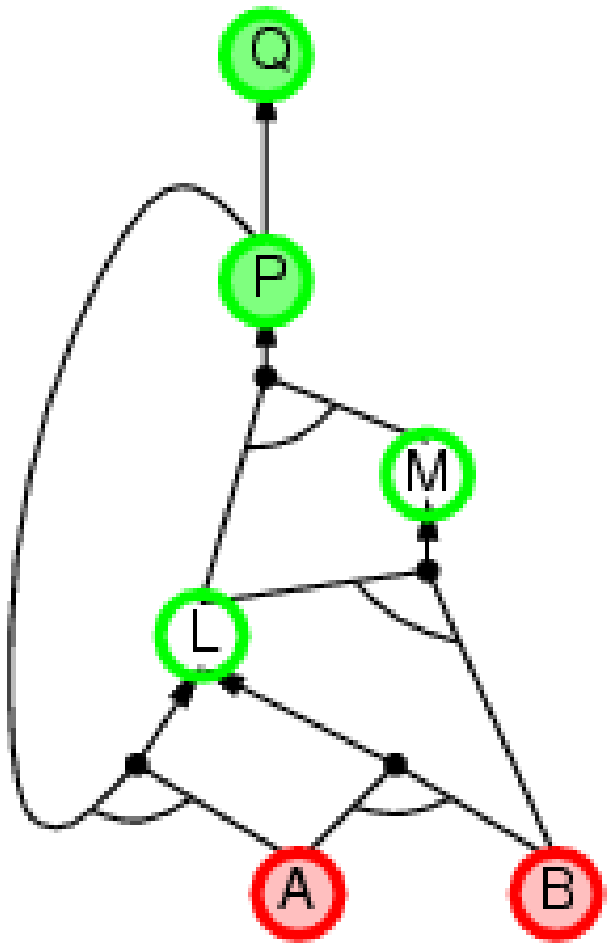
Backward chaining example



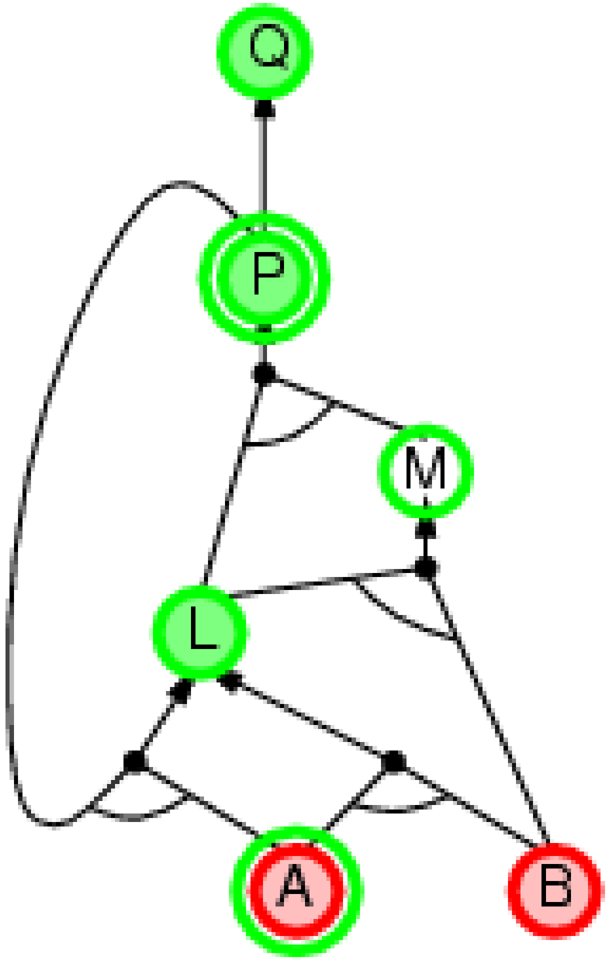
Backward chaining example



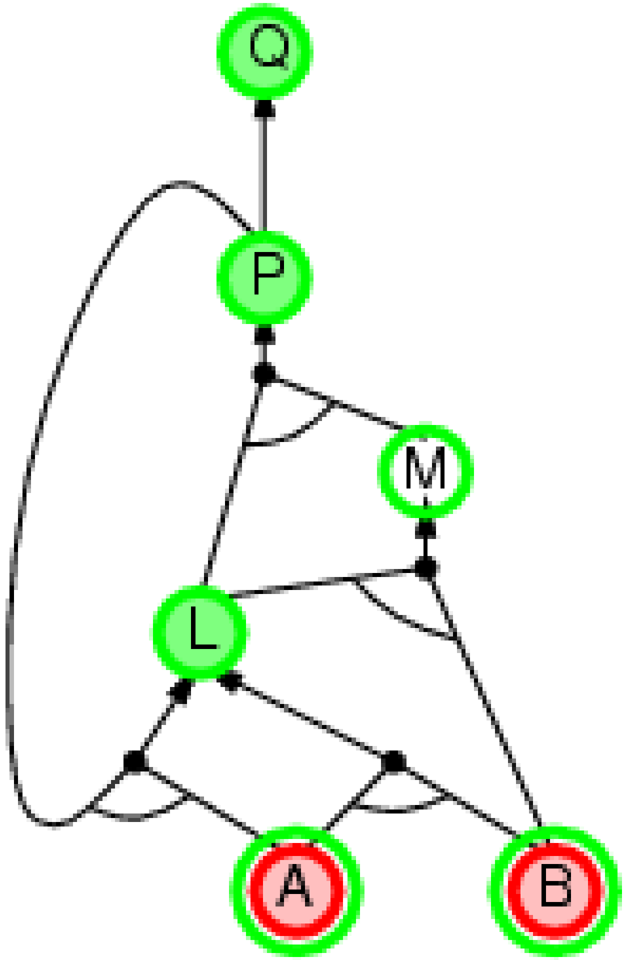
Backward chaining example



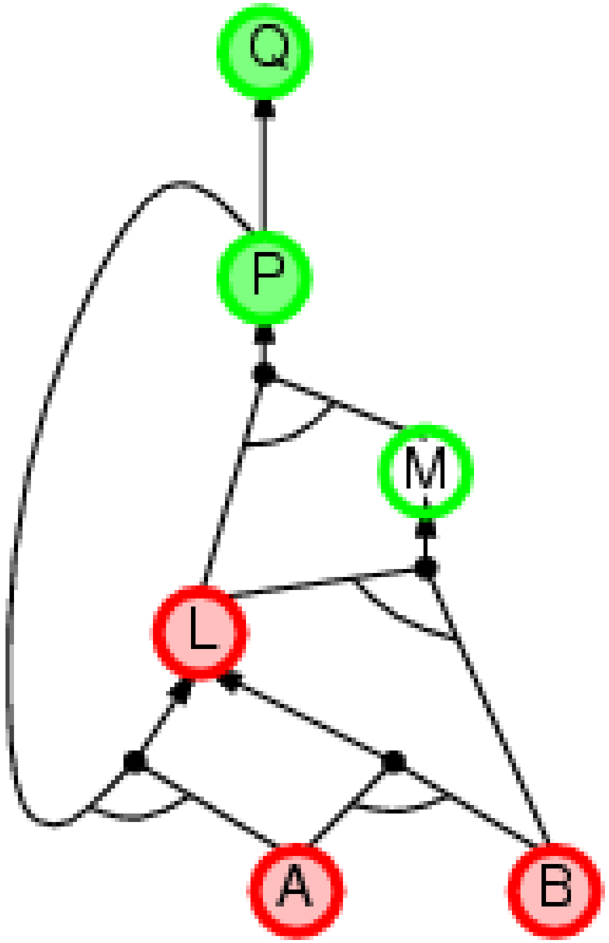
Backward chaining example



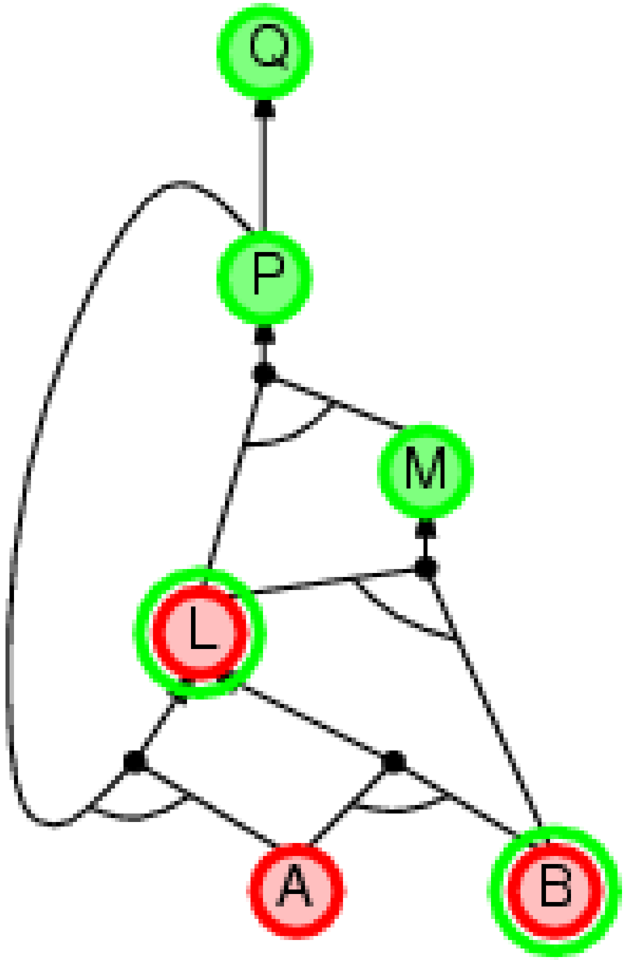
Backward chaining example



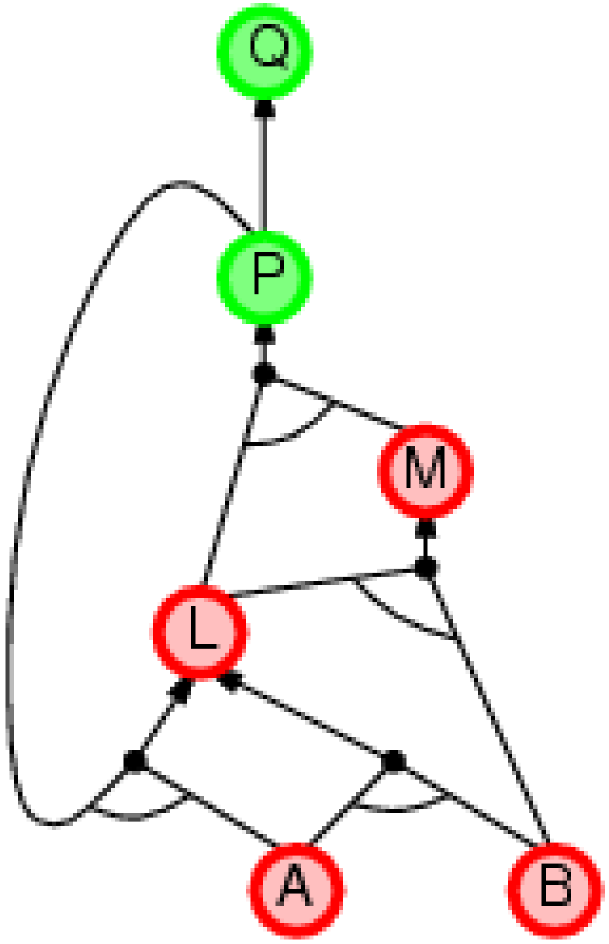
Backward chaining example



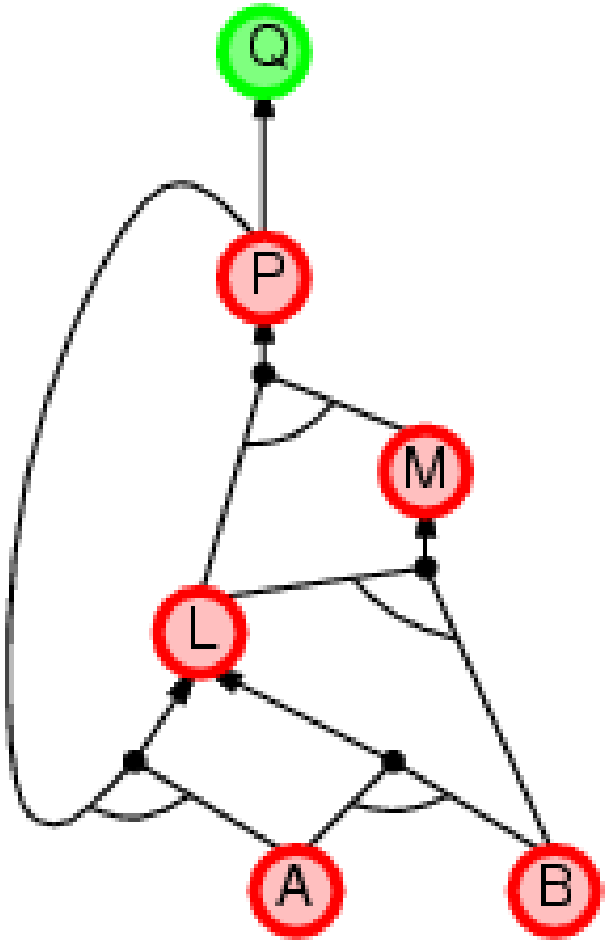
Backward chaining example



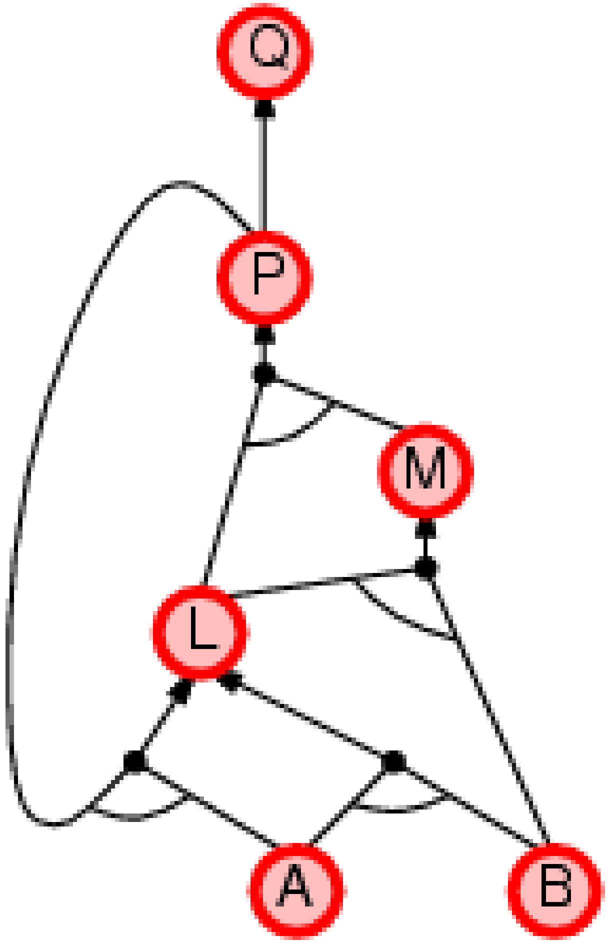
Backward chaining example



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? How do I get into a PhD program?
- Complexity of BC can be much less than linear in size of KB

Resolution

Conjunctive Normal Form (CNF)

conjunction of disjunctions of literals

clauses

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

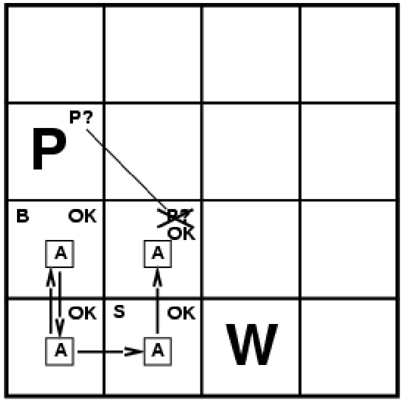
- Resolution inference rule (for CNF):
-

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

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



Resolution

Soundness of resolution inference rule:

$$\frac{\begin{array}{c}\neg(l_1 \vee \ldots \vee l_{i-1} \vee l_{i+1} \vee \ldots \vee l_k) \Rightarrow l_i \\ \neg m_j \Rightarrow (m_1 \vee \ldots \vee m_{j-1} \vee m_{j+1} \vee \ldots \vee m_n)\end{array}}{\neg(l_1 \vee \ldots \vee l_{i-1} \vee l_{i+1} \vee \ldots \vee l_k) \Rightarrow (m_1 \vee \ldots \vee m_{j-1} \vee m_{j+1} \vee \ldots \vee m_n)}$$

Conversion to CNF

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

1. Eliminate $\Leftrightarrow$, replacing $\alpha \Leftrightarrow \beta$ with $(\alpha \Rightarrow \beta)\land(\beta \Rightarrow \alpha)$.
2.

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

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:

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

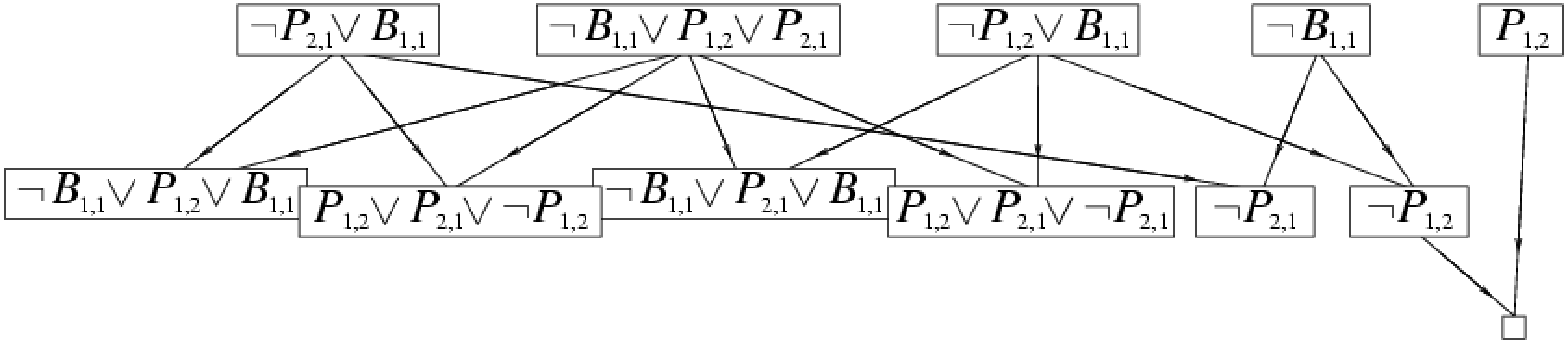
Resolution algorithm

- 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 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

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



Resolution strategies (heuristics for clause selection)

1. **Unit clause preference**: P, $\neg$P $\vee$ [.....] ==> [.....] shorter!

2. **'Set of Support'**

   resolution (a clause from a 'Set of Support' and an external clause), rezolvent into 'Set of Support'-ba,

   complete, if clauses not in 'Set of Support' are satisfiable

   in practice: 'Set of Support' = the negated question (the rest is assumed to be true)

3. **Input resolution**

   The resolvent in step i. is one of the clause in step i+1 (it starts with the question). **Complete in Horn KBs.**

4. **Linear resolution**

   P and Q can be resolved, if P is in the KB or P is the ancestor of Q in the proof tree. **Complete.**

5. **Pruning**

   Eliminate all rules more specific than in the knowledge base.

Summary

- Truth and proofs
- The „truth-table method" for validity&soundness
- Automated reasoning
  - Forward chaining, Backward chaining
    - linear-time, complete for Horn clauses
  - Resolution
    - Conjunctive normal form (CNF)
    - Inference step
      - Equivalence with if-then forms („transitivity")
      - Complexity preserving (cf. Modus Ponens)
      - Covers Modus Ponens(!, unit clause)
    - Framework
      - proof by refutation, reductio ad absurdum
      - Heuristics: resolution strategy
    - Complete for propositional logic