Logical agents

AIMA.3rd Chapter 7.4-6

Outline

◇ Propositional (Boolean) logic

◇ Equivalence, validity, satisfiability

◇ Inference rules and theorem proving
  - forward chaining
  - backward chaining
  - resolution

Propositional logic: Syntax

Propositional logic is the simplest logic—illustrates basic ideas

The syntax of propositional logic defines the allowable sentences.

The atomic sentences consist of a single proposition symbol, e.g., P_1, P_2.

A literal is either an atomic sentence (a positive literal) or a negated atomic sentence (a negative literal)

Propositional logic: Syntax

Complex sentences are constructed from simpler sentences, using parentheses and logical connectives.

There are five connectives in common use:

– $\neg$ (not): If S is a sentence, $\neg S$ is a sentence (negation)

– $\wedge$ (and): If S_1 and S_2 are sentences, $S_1 \wedge S_2$ is a sentence (conjunction)

– $\vee$ (or): If S_1 and S_2 are sentences, $S_1 \vee S_2$ is a sentence (disjunction)

– $\Rightarrow$ (implies): A sentence such as $(W_{1,3} \wedge P_{3,1}) \Rightarrow \neg W_{2,2}$
are called implication (a.k.a. rules or if-then statements)
Its premise is $(W_{1,3} \wedge P_{3,1})$, and its conclusion is $\neg W_{2,2}$.

– $\Leftrightarrow$ (iff): If S_1 and S_2 are sentences, $S_1 \Leftrightarrow S_2$ is a sentence
called (biconditional)

Propositional logic: Semantics

The semantics defines the rules for determining the truth of a sentence with respect to a particular model.
In propositional logic, a model fixes the truth value *true* or *false* for every proposition symbol

Rules for evaluating truth with respect to a model m:

$\neg S$	is true iff	S	is false		
$S_1 \wedge S_2$	is true iff	S_1	is true **and**	S_2	is true
$S_1 \vee S_2$	is true iff	S_1	is true **or**	S_2	is true
$S_1 \Rightarrow S_2$	is true iff	S_1	is false **or**	S_2	is true
i.e.,	is false iff	S_1	is true **and**	S_2	is false
$S_1 \Leftrightarrow S_2$	is true iff	$S_1 \Rightarrow S_2$	is true **and**	$S_2 \Rightarrow S_1$	is true

Truth tables for connectives

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*

* Truth table for $P \Rightarrow Q$ may not quite fit ones intuitive understanding of "P implies Q" or "if P then Q."
You can think of the $P \Rightarrow Q$ as "If P is true, then I am claiming that Q is true. Otherwise I am making no claim."

* Propositional logic does not require any relation of *causation* or *relevance* between P and Q.

Propositional logic: example semantics

E.g. $m_1 = \begin{array}{ccc} P_{1,2} & P_{2,2} & P_{3,1} \\ true & true & false \end{array}$

(With these symbols, 8 possible models, can be enumerated automatically.)

Simple recursive process evaluates an arbitrary sentence, e.g.,

$\neg P_{1,2} \wedge (P_{2,2} \vee P_{3,1}) = true \wedge (false \vee true) = true \wedge true = true$

Wumpus world sentences

Let $P_{i,j}$ be true if there is a pit in $[i, j]$.
Let $B_{i,j}$ be true if there is a breeze in $[i, j]$.

"There is no pit in [1,1]"

R_1: $\neg P_{1,1}$

"Pits cause breezes in adjacent squares"

$R_2 : B_{1,1} \;\Leftrightarrow\; (P_{1,2} \lor P_{2,1})$
$R_3 : B_{2,1} \;\Leftrightarrow\; (P_{1,1} \lor P_{2,2} \lor P_{3,1})$

"A square is breezy **if and only if** there is an adjacent pit"

$R_4 : \neg B_{1,1}$
$R_5 : B_{2,1}$

Model checking vs theorem proving

Model checking enumerates all models and showing that the sentence must hold in all models.

Theorem proving applys rules of inference directly to the sentences in KB to construct a proof of the desired sentence without consulting models.

If the number of models is large but the length of the proof is short, then theorem proving is more efficient than model checking.

Truth tables for inference

Goal of inference now is to decide whether $KB \models \alpha$ for some sentence α.
Truth table inference: Enumerate the models (truth assignment to every proposition symbols), and check that α is true in every model in which KB is true.

$B_{1,1}$	$B_{2,1}$	$P_{1,1}$	$P_{1,2}$	$P_{2,1}$	$P_{2,2}$	$P_{3,1}$	R_1	R_2	R_3	R_4	R_5	KB
false	*false*	*false*	*false*	*false*	*false*	*false*	*true*	*true*	*true*	*true*	*false*	*false*
false	*false*	*false*	*false*	*false*	*false*	*true*	*true*	*true*	*false*	*true*	*false*	*false*
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
false	*true*	*false*	*false*	*false*	*false*	*false*	*true*	*true*	*false*	*true*	*true*	*false*
false	*true*	*false*	*false*	*false*	*false*	*true*	*true*	*true*	*true*	*true*	*true*	*true*
false	*true*	*false*	*false*	*false*	*true*	*false*	*true*	*true*	*true*	*true*	*true*	*true*
false	*true*	*false*	*false*	*false*	*true*	*true*	*true*	*true*	*true*	*true*	*true*	*true*
false	*true*	*false*	*false*	*true*	*false*	*false*	*true*	*false*	*false*	*true*	*true*	*false*
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
true	*true*	*true*	*true*	*true*	*true*	*true*	*false*	*true*	*true*	*false*	*true*	*false*

Inference by enumeration

The following depth-first truth-table enumeration algorithm for deciding propositional entailment is sound and complete

```
function TT-ENTAILS?(KB, α) returns true or false
   inputs: KB, the knowledge base, a sentence in propositional logic
           α, the query, a sentence in propositional logic

   symbols ← a list of the proposition symbols in KB and α
   return TT-CHECK-ALL(KB, α, symbols, [ ])
```

```
function TT-CHECK-ALL(KB, α, symbols, model) returns true or false
   if EMPTY?(symbols) then
      if PL-TRUE?(KB, model) then return PL-TRUE?(α, model)
      else return true
   else do
      P ← FIRST(symbols); rest ← REST(symbols)
      return TT-CHECK-ALL(KB, α, rest, EXTEND(P, true, model)) and
                TT-CHECK-ALL(KB, α, rest, EXTEND(P, false, model))
```

PL-TRUE? return *true* if a sentence holds within a model. The variable *model* represents a partial model. **and** is a logical operation on its two arguments, returning *true* or *false*.

Inference by enumeration: Model checking

The TT-ENTAILS? algorithm is sound because it implements directly the definition of entailment,
and complete because it works for any KB and α and always terminates there are only finitely many models to examine.

Time complexity is $O(2^n)$ when KB and α contain n symbols; problem is **co-NP-complete**

Concepts in theorem proving: Logical equivalence

Two sentences are logically equivalent iff they are true in the same set of models:

$\alpha \equiv \beta$ if and only if $\alpha \models \beta$ and $\beta \models \alpha$

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

Concepts cont.: Validity

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

e.g., $True$, $A \vee \neg A$, $A \Rightarrow A$, $(A \wedge (A \Rightarrow B)) \Rightarrow B$

Validity is connected to inference via the Deduction Theorem:

For any sentence α and β, $\alpha \models \beta$ iff $(\alpha \Rightarrow \beta)$ is valid

From this, we can decide if $\alpha \models \beta$ by

checking that $(\alpha \Rightarrow \beta)$ is $True$ in every model (TT inference)

by proving $(\alpha \Rightarrow \beta)$ is $True$.

Concepts cont.: Satisfiability

A sentence is satisfiable if it is true in **some** model
e.g., $A \vee B$, C

A sentence is unsatisfiable if it is true in **no** models
e.g., $A \wedge \neg A$

Validity and satisfiability are connected:
- α is valid iff $\neg\alpha$ is unsatisfiable
- α is satisfiable iff $\neg\alpha$ is not valid.

Satisfiability is connected to inference via the following:
$\alpha \models \beta$ if and only if $(\alpha \wedge \neg\beta)$ is unsatisfiable
i.e., proof by contradiction: One assumes a sentence β to be $false$ and shows that this leads to a contradiction with known axioms α ($(\alpha \wedge \neg\beta)$).

* Note: The problem of determining the satisfiability of sentences in propositional logic the SAT problem was the first problem proved to be NP-complete.

Proof methods summary

Proof methods divide into (roughly) two kinds:

Theorem proving: 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.

– Typically require translation of sentences into a normal form

Model checking: Enumeration of models

truth table enumeration (always exponential in n)

improved backtracking, e.g., Davis–Putnam–Logemann–Loveland

heuristic search in model space (sound but incomplete)

  e.g., min-conflicts-like hill-climbing algorithms

Forward and backward chaining

Horn Form (restricted)

KB = **conjunction** of **Horn clauses**

Horn clause =

- ♢ disjunction of literals of which at most one is positive; 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

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

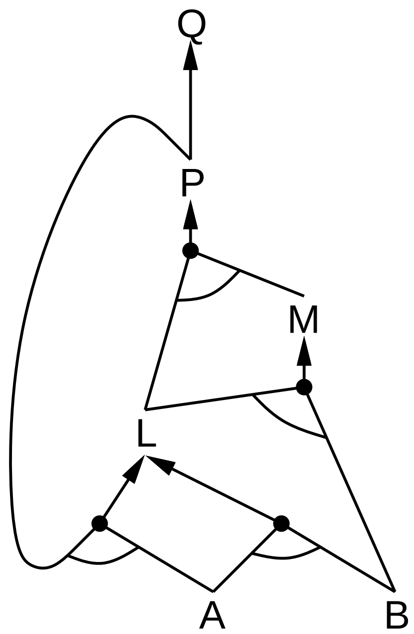
("whenever any sentences of the form $\alpha_1, \ldots, \alpha_n$, and $\alpha_1 \wedge \cdots \wedge \alpha_n \Rightarrow \beta$ are given, then the sentence β can be inferred.") Can be used with forward chaining or backward chaining.

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

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

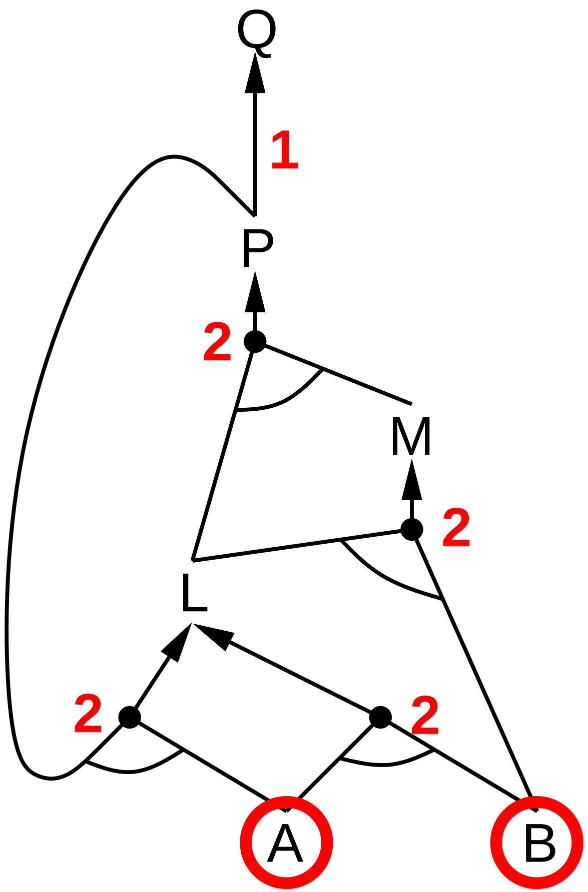


Goal: determines if a proposition symbol q (i.e. query) is entailed by a KB

Forward chaining algorithm

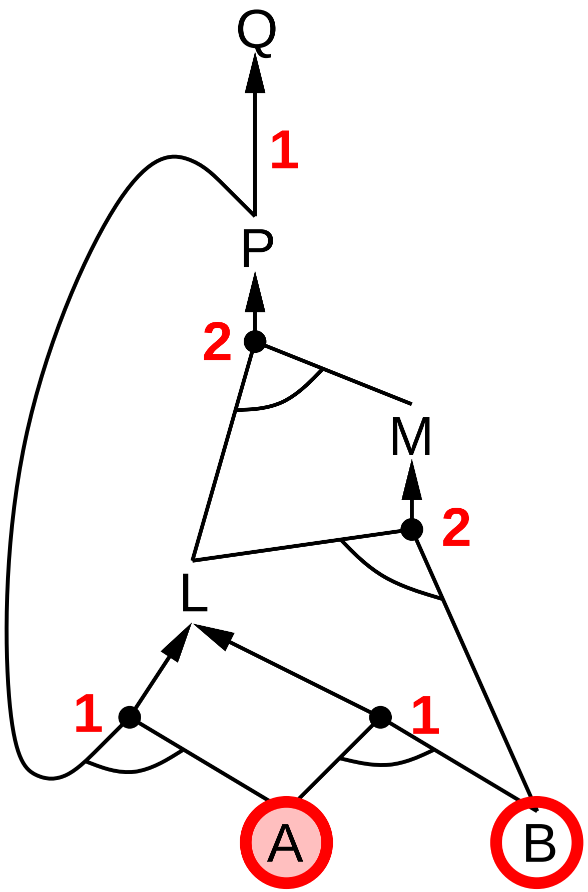
```
function PL-FC-Entails?(KB, q) returns true or false
   inputs: KB, the knowledge base, a set of propositional Horn clauses
           q, the query, a proposition symbol
  local variables: count, a table, indexed by clause, initially the number of premises
                   inferred, a table, indexed by symbol, each entry initially false
                   agenda, a list of symbols, initially the symbols known in KB

   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 example



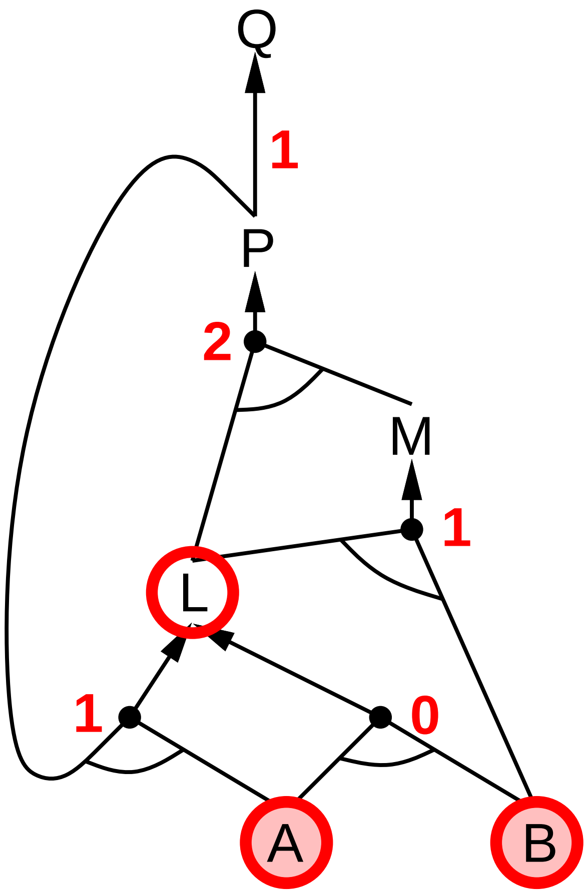
It begins from known facts (positive literals) in the knowledge base.

Forward chaining example



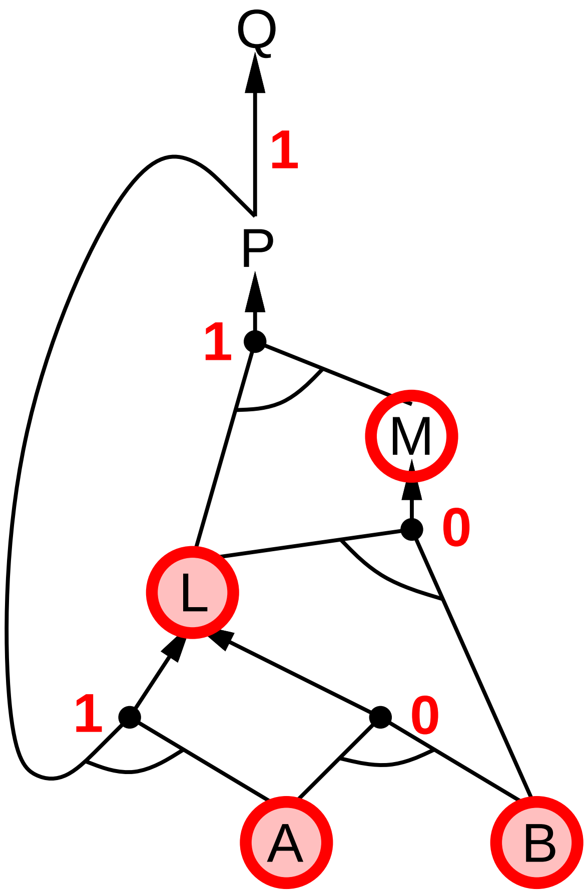
If all the premises of an implication are known, then its conclusion is added to the set of known facts.

Forward chaining example



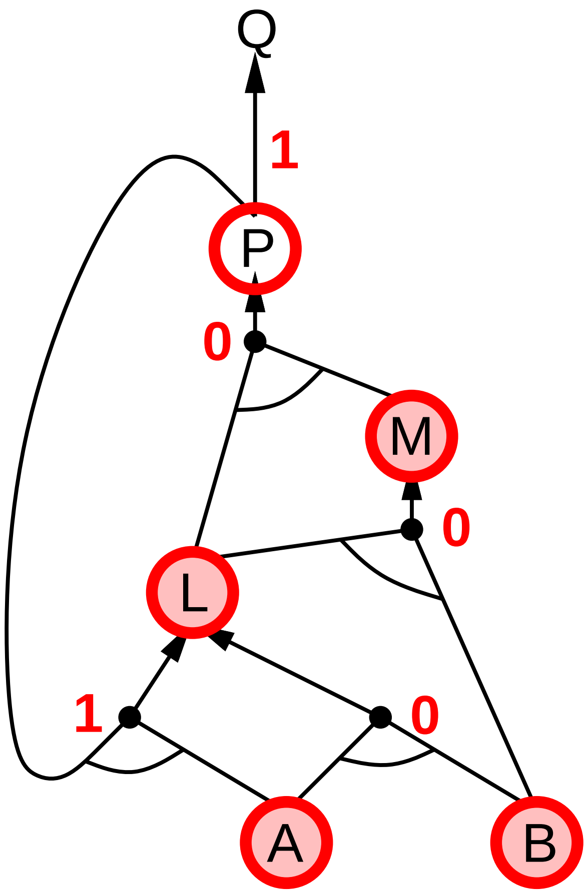
If all the premises of an implication are known, then its conclusion is added to the set of known facts.

Forward chaining example



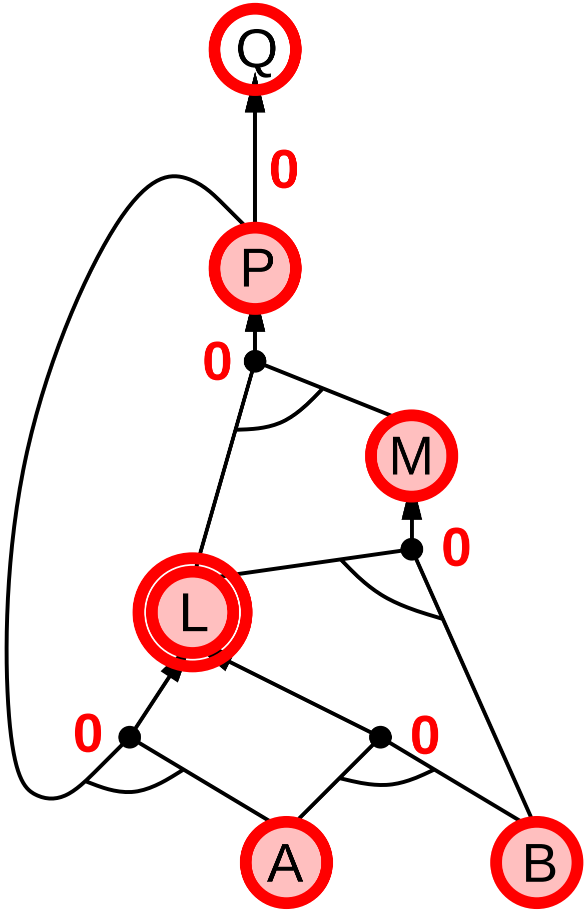
The known leaves (here, A and B) are set, and inference propagates up the graph as far as possible.

Forward chaining example

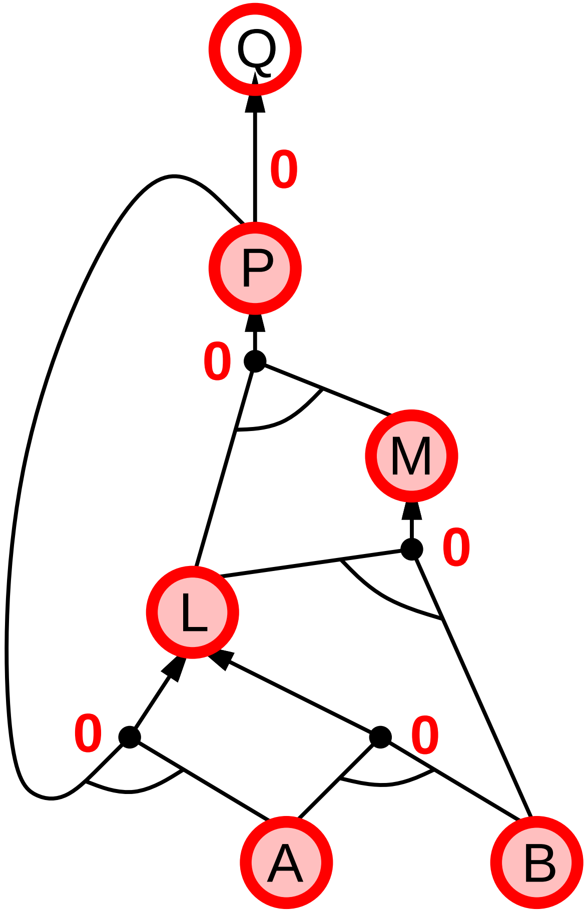


Wherever a conjunction appears, the propagation waits until all the conjuncts are known before proceeding.

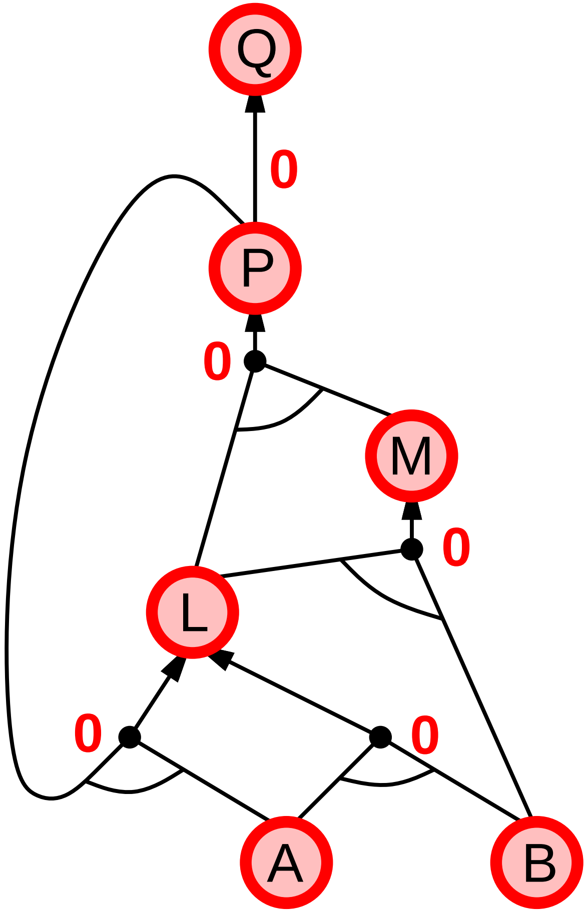
Forward chaining example



Forward chaining example



Forward chaining example



FC is **sound**: every inference is essentially an application of *Modus Ponens*

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 $true$ to each symbols inferred during the FC and $false$ for all other symbols

3. Every clause in the original KB is true in m

   Proof by contradiction: Suppose a clause $a_1 \wedge \ldots \wedge a_k \Rightarrow b$ is false in m

   Then $a_1 \wedge \ldots \wedge a_k$ is true in m and b is false in m

   This contradicts our assumption that the algorithm has reached a 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

General idea: construct any model of KB by sound inference, check α

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

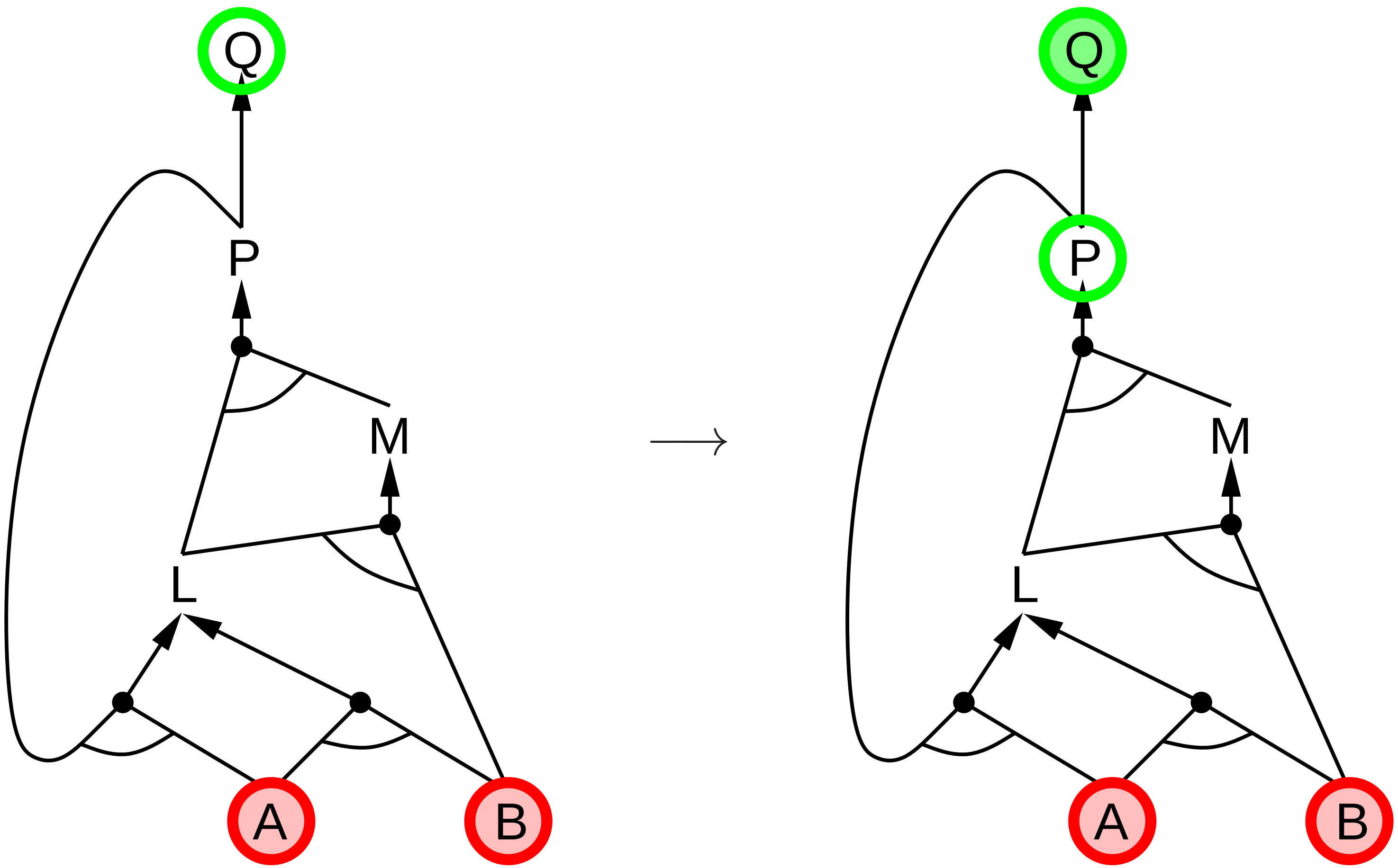
Avoid loops: check if new subgoal is already on the goal stack

Avoid repeated work: check if new subgoal

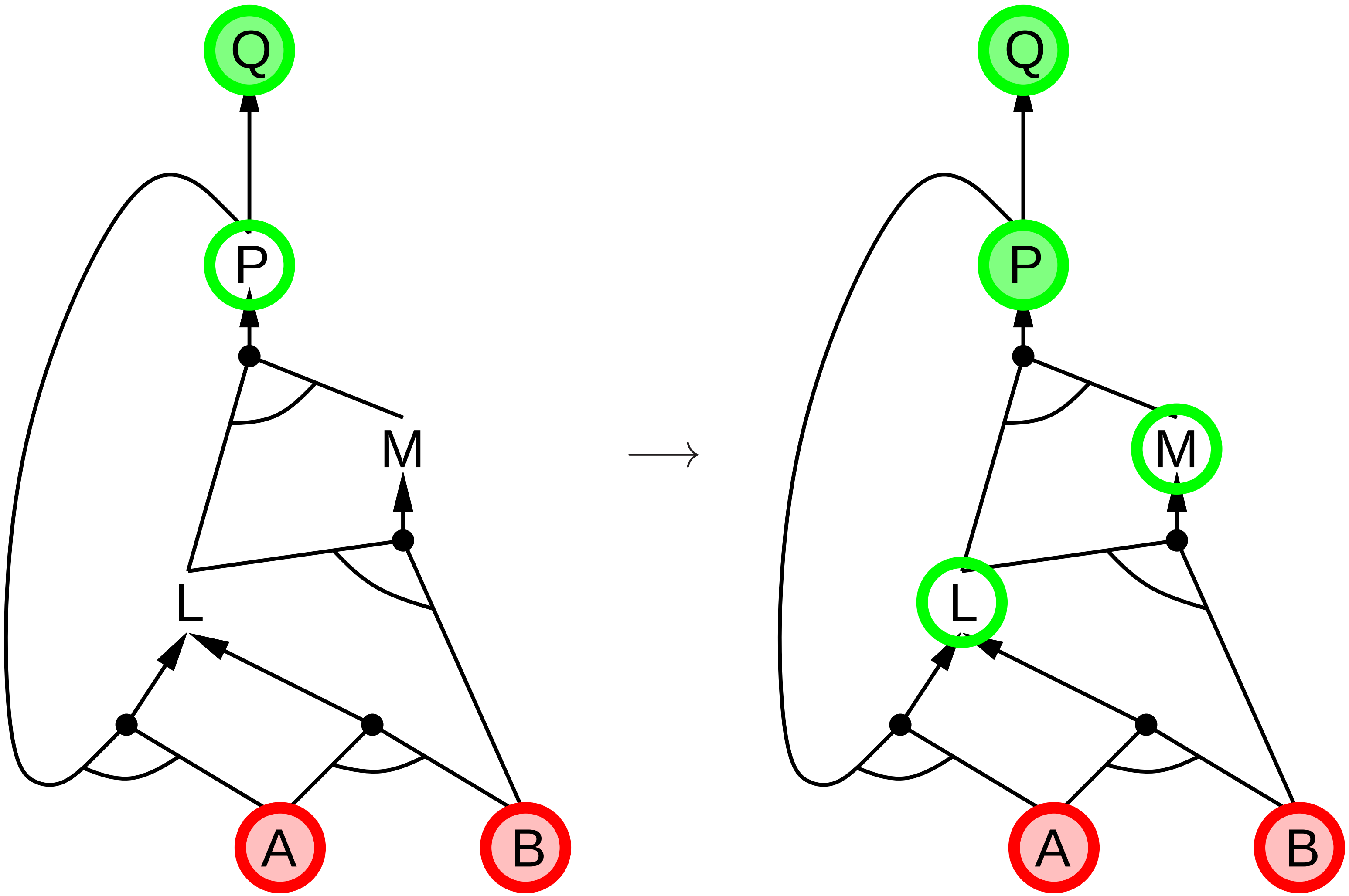
1) has already been proved true, or
2) has already failed

The algorithm is essentially identical to the And-Or-Graph-Search algorithm

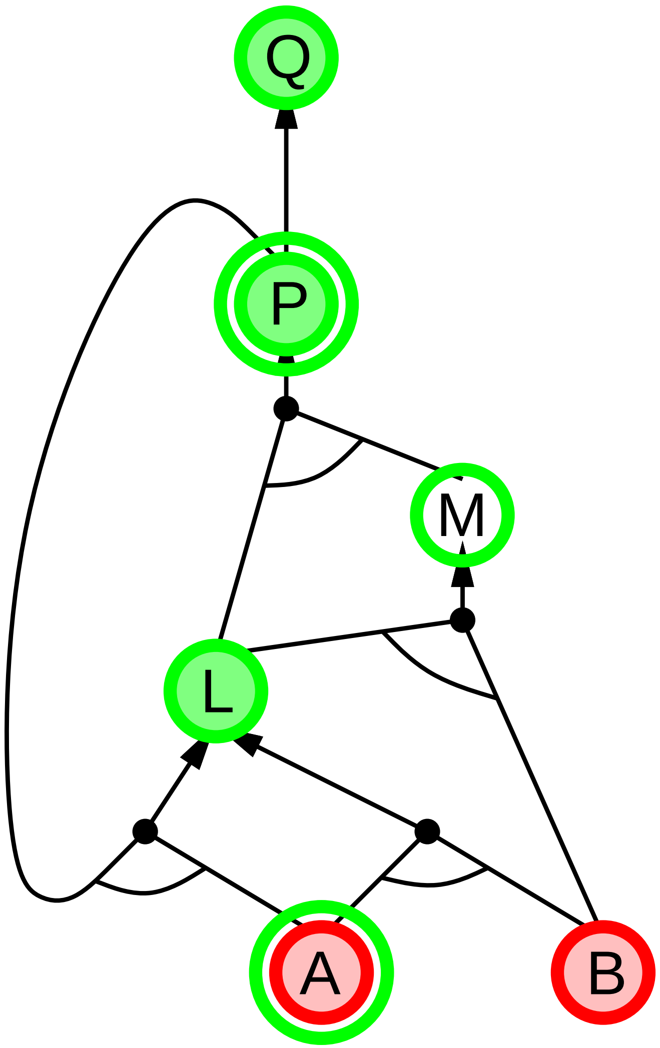
Backward chaining example



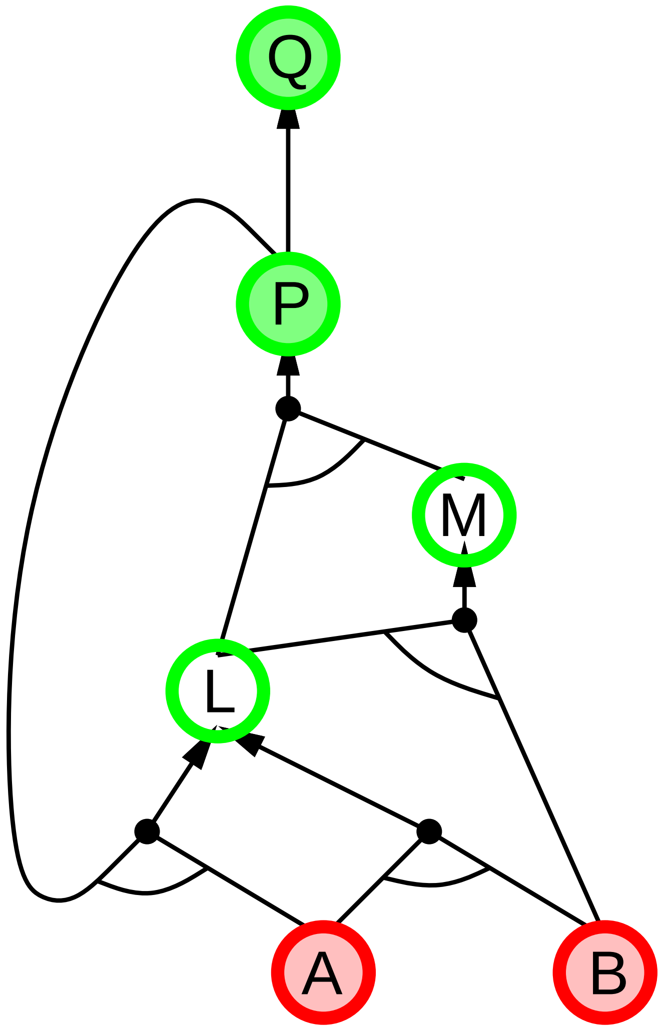
Backward chaining example



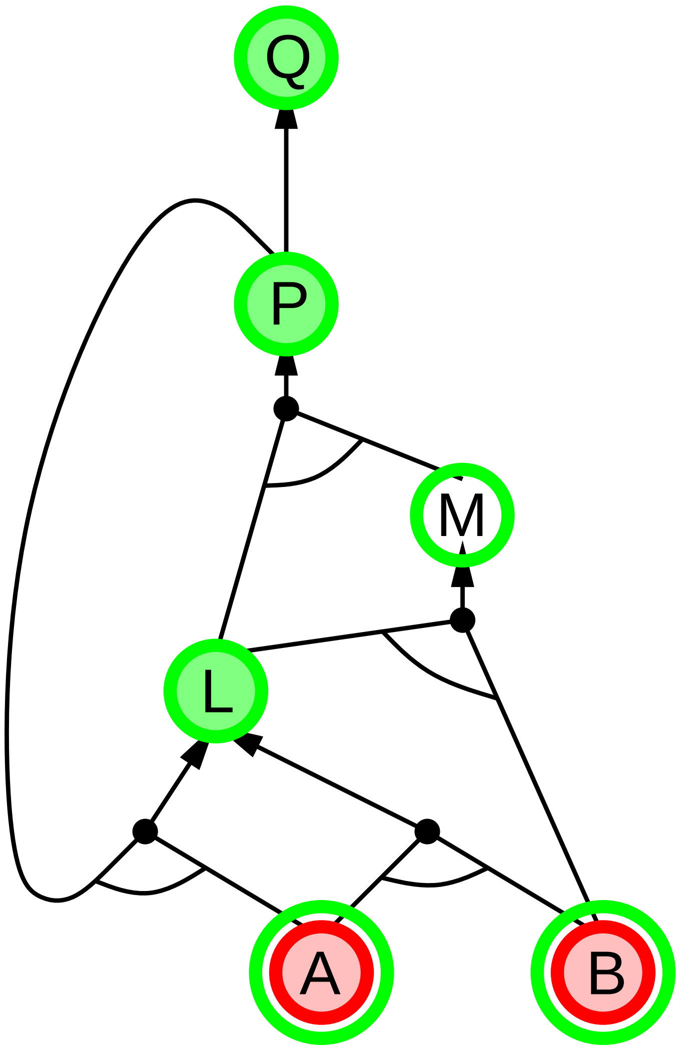
Backward chaining example



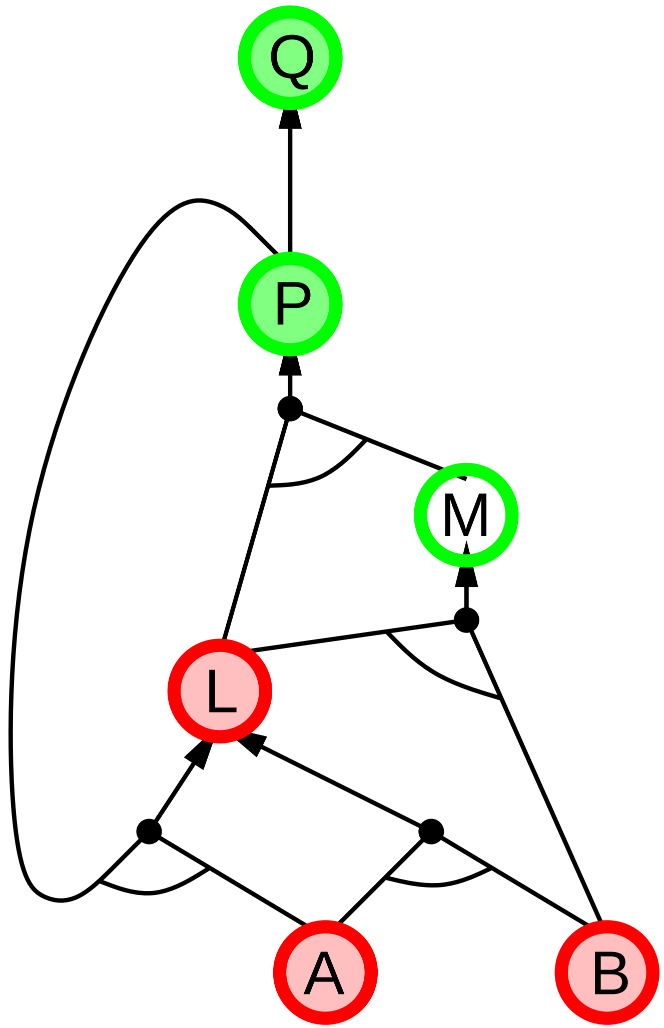
$\longrightarrow$



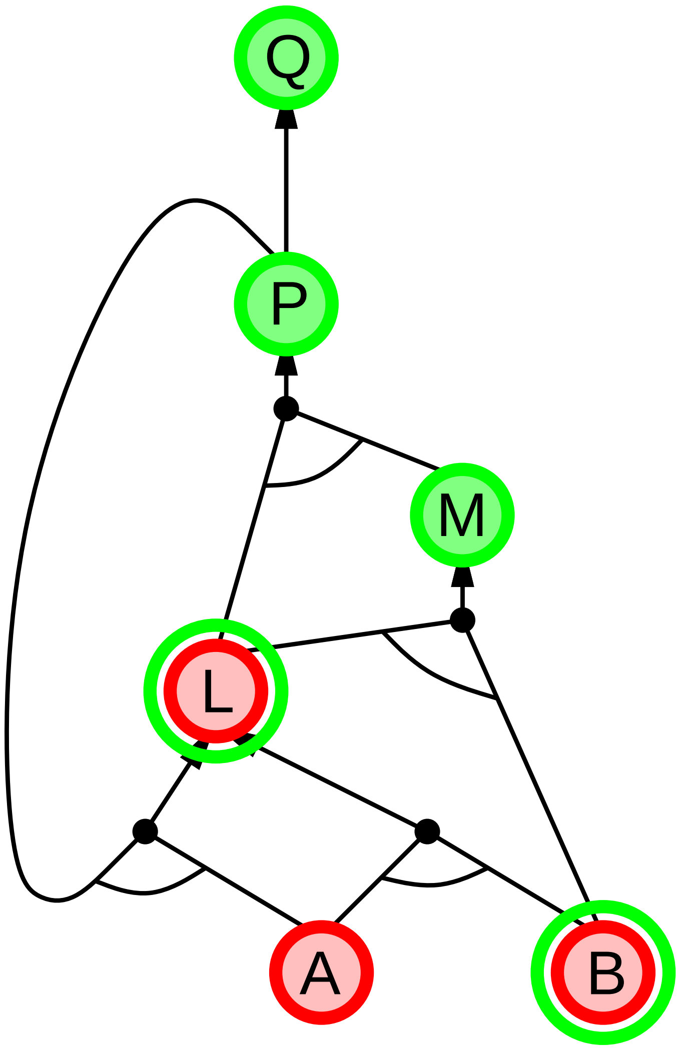
Backward chaining example



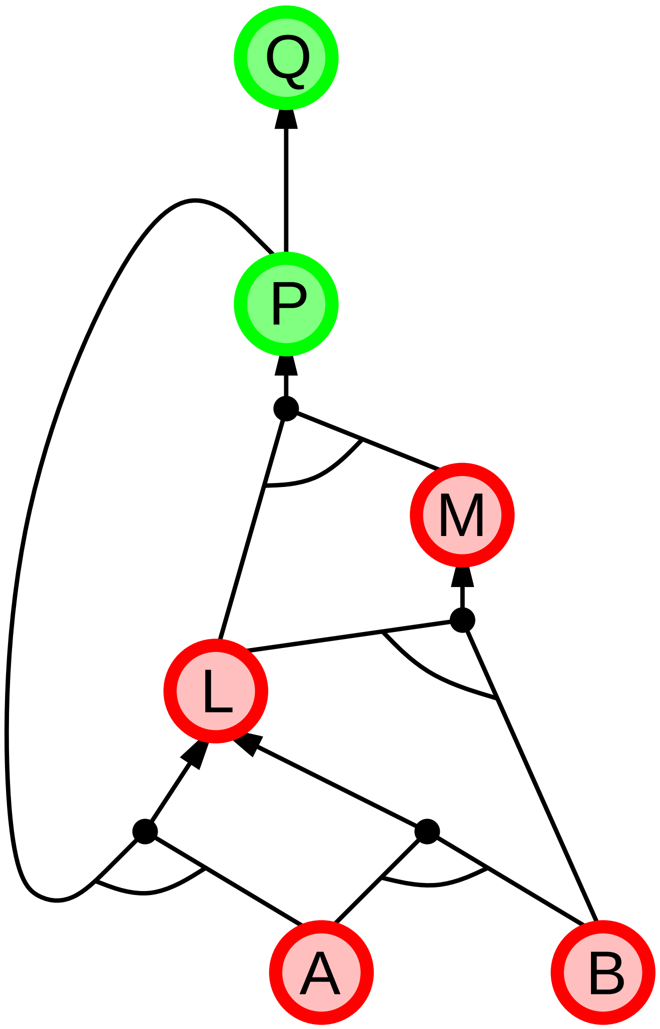
$\longrightarrow$



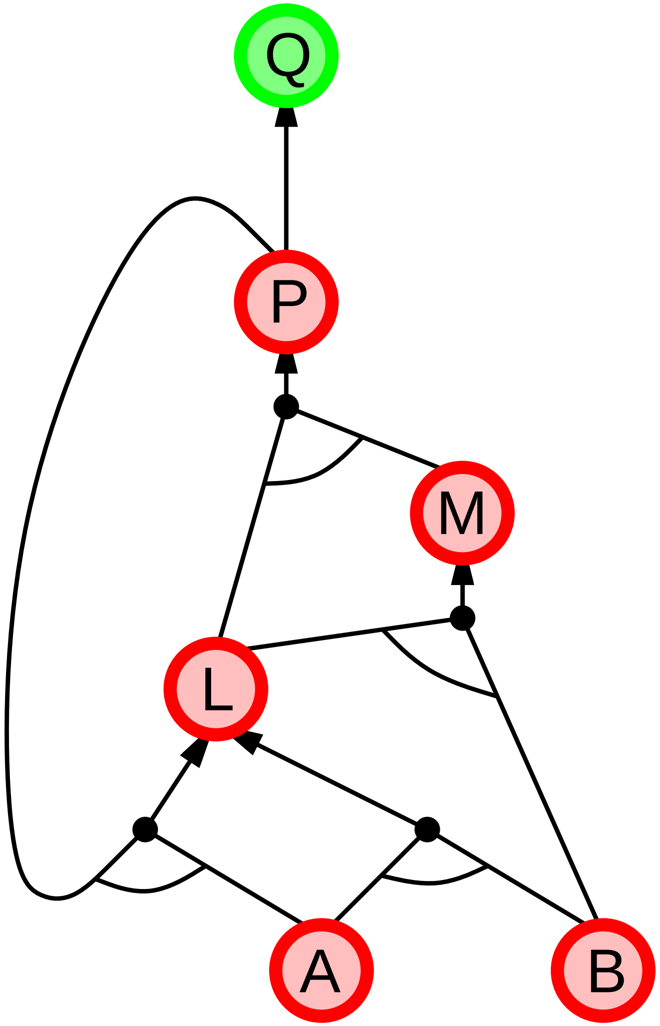
Backward chaining example



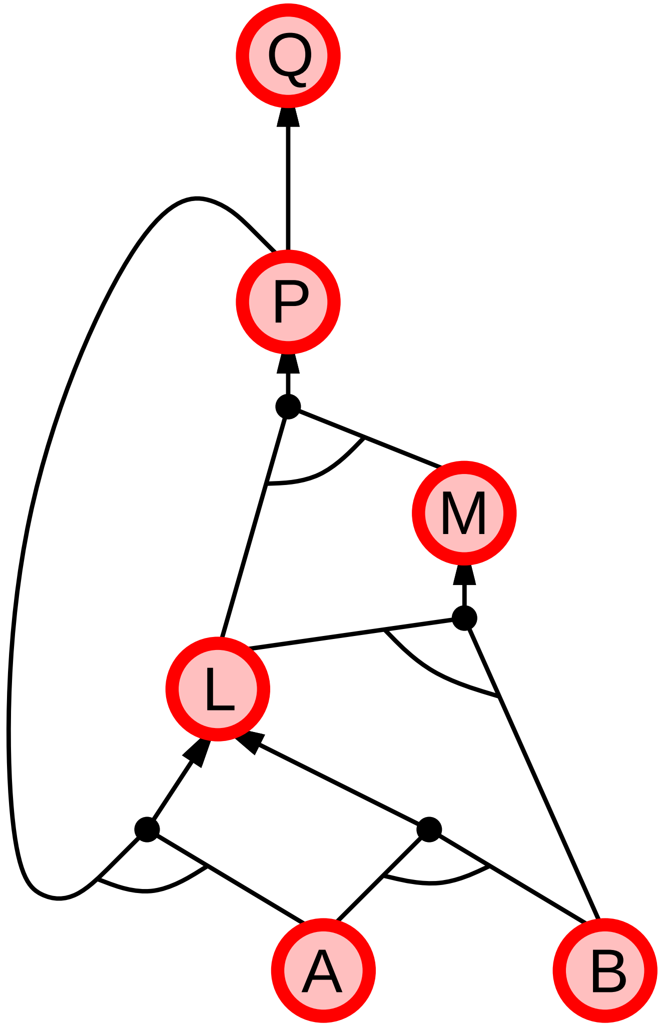
$\longrightarrow$



Backward chaining example



$\longrightarrow$



Forward vs. backward chaining

FC is data-driven, cf. automatic, unconscious processing,
e.g., object recognition, routine decisions

May do lots of work that is irrelevant to the goal:
an efficient implementation runs in linear time.

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
the process touches only relevant facts.

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.

Forward, backward chaining are linear-time, complete for Horn clauses
Resolution is complete for propositional logic

Propositional logic lacks expressive power