

# PROPOSITIONAL LOGIC<sup>1</sup>

## LECTURE 2

---

<sup>1</sup>The slides have been prepared using the textbook material available on the web, and the slides of the previous editions of the course by Prof. Luigia Carlucci Aiello

## Summary

- ◇ Propositional Logic Russell & Norvig Sec. 7.4 - 7.5
- ◇ Inference in propositional logic Russell & Norvig Sec. 7.6 part

Propositional logic is the simplest logic—illustrates basic approach to knowledge representation

## Propositional logic: Syntax

*True* and *False* are propositional symbols

Other propositional symbols are denoted as  $P_1, P_2, \dots$

If  $S$  is a propositional symbol,  $S$  is a sentence

If  $S$  is a sentence,  $\neg S$  is a sentence (**negation**)

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

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

If  $S_1$  and  $S_2$  are sentences,  $S_1 \Rightarrow S_2$  is a sentence (**implication**)

If  $S_1$  and  $S_2$  are sentences,  $S_1 \Leftrightarrow S_2$  is a sentence (**biconditional**)

# Propositional logic: Notation

Alternatives to R&N notation:

$\supset$  for  $\Rightarrow$  ;

$\equiv$  for  $\Leftrightarrow$  ;

0, 1 for *True*, *False*

## 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  $[i, j]$ .

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

$$\neg B_{1,1}$$

$$B_{2,1}$$

## Propositional logic: Semantics

The truth of a sentence can be determined, given an interpretation  $m$ , that assigns a truth value to every propositional symbol:

*True* is true

*False* is false

$P$  is true iff  $P$  is true in  $m$

$P$  is false iff  $P$  is false in  $m$

## Propositional logic: Semantics

Rules for evaluating truth of complex sentences:

$\neg S$ 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

## Propositional logic: Semantics

Each interpretation specifies true/false for each proposition symbol

E.g.  $P_{1,2}$   $P_{2,2}$   $P_{3,1}$   
*false true false*

A simple recursive process evaluates an arbitrary sentence, e.g.,

$$\begin{aligned}\neg P_{1,2} \wedge (P_{2,2} \vee P_{3,1}) &= \\ \textit{true} \wedge (\textit{false} \vee \textit{true}) &= \\ \textit{true} \wedge \textit{true} &= \\ \textit{true} &\end{aligned}$$



## Truth tables for connectives

$P$	$Q$	$\neg P$	$P \wedge Q$	$P \vee Q$	$P \Rightarrow Q$	$P \Leftrightarrow Q$
<i>false</i>	<i>false</i>	<i>true</i>	<i>false</i>	<i>false</i>	<i>true</i>	<i>true</i>
<i>false</i>	<i>true</i>	<i>true</i>	<i>false</i>	<i>true</i>	<i>true</i>	<i>false</i>
<i>true</i>	<i>false</i>	<i>false</i>	<i>false</i>	<i>true</i>	<i>false</i>	<i>false</i>
<i>true</i>	<i>true</i>	<i>false</i>	<i>true</i>	<i>true</i>	<i>true</i>	<i>true</i>

## Wumpus world KB

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  $[i, j]$ .

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

$$\neg B_{1,1}$$

$$B_{2,1}$$

“Pits cause breezes in adjacent squares”

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

$$B_{2,1} \Leftrightarrow (P_{1,1} \vee P_{2,2} \vee P_{3,1})$$

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

# Entailment

Entailment:

$$KB \models \alpha$$

Knowledge base  $KB$  entails sentence  $\alpha$

if and only if

$\alpha$  is true in all worlds where  $KB$  is true

$$KB \models \alpha \text{ if and only if } M(KB) \subseteq M(\alpha)$$

# Truth tables for inference

$B_{1,1}$	$B_{2,1}$	$P_{1,1}$	$P_{1,2}$	$P_{2,1}$	$P_{2,2}$	$P_{3,1}$	$KB$	$\neg P_{1,2}$
<i>false</i>	<i>false</i>	<i>false</i>	<i>false</i>	<i>false</i>	<i>false</i>	<i>false</i>	<i>false</i>	<i>true</i>
<i>false</i>	<i>false</i>	<i>false</i>	<i>false</i>	<i>false</i>	<i>false</i>	<i>true</i>	<i>false</i>	<i>true</i>
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
<i>false</i>	<i>true</i>	<i>false</i>	<i>false</i>	<i>false</i>	<i>false</i>	<i>false</i>	<i>false</i>	<i>true</i>
<i>false</i>	<i>true</i>	<i>false</i>	<i>false</i>	<i>false</i>	<i>false</i>	<i>true</i>	<u><i>true</i></u>	<u><i>true</i></u>
<i>false</i>	<i>true</i>	<i>false</i>	<i>false</i>	<i>false</i>	<i>true</i>	<i>false</i>	<u><i>true</i></u>	<u><i>true</i></u>
<i>false</i>	<i>true</i>	<i>false</i>	<i>false</i>	<i>false</i>	<i>true</i>	<i>true</i>	<u><i>true</i></u>	<u><i>true</i></u>
<i>false</i>	<i>true</i>	<i>false</i>	<i>false</i>	<i>true</i>	<i>false</i>	<i>false</i>	<i>false</i>	<i>true</i>
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
<i>true</i>	<i>true</i>	<i>true</i>	<i>true</i>	<i>true</i>	<i>true</i>	<i>true</i>	<i>false</i>	<i>false</i>

# Checking $\models$ by model enumeration

## Depth-first enumeration of all models

```
function TT-ENTAILS?( $KB, \alpha$ ) returns true – false
```

```
   $symbols \leftarrow$  in  $KB$  e  $\alpha$ 
```

```
  return TT-CHECK-ALL( $KB, \alpha, symbols, []$ )
```

---

```
function TT-CHECK-ALL( $KB, \alpha, symbols, model$ ) returns true/false
```

```
  if EMPTY?( $symbols$ ) then
```

```
    if PL-TRUE?( $KB, model$ ) then return PL-TRUE?( $\alpha, model$ )
```

```
    else return true
```

```
  else do
```

```
     $P \leftarrow$  FIRST( $symbols$ );  $rest \leftarrow$  REST( $symbols$ )
```

```
    return TT-CHECK-ALL( $KB, \alpha, rest, EXTEND(P, true, model)$ )
```

```
      and
```

```
      TT-CHECK-ALL( $KB, \alpha, rest, EXTEND(P, false, model)$ )
```

$O(2^n)$  for  $n$  symbols; problem is co-NP-complete

## Logical equivalence

Two sentences are **logically equivalent** iff true in same models:

$$\alpha \equiv \beta \quad \text{if and only if} \quad \alpha \models \beta \text{ and } \beta \models \alpha$$

## Logical equivalence table

$(\alpha \wedge \beta) \equiv (\beta \wedge \alpha)$	commutativity of $\wedge$
$(\alpha \vee \beta) \equiv (\beta \vee \alpha)$	commutativity of $\vee$
$((\alpha \wedge \beta) \wedge \gamma) \equiv (\alpha \wedge (\beta \wedge \gamma))$	associativity of $\wedge$
$((\alpha \vee \beta) \vee \gamma) \equiv (\alpha \vee (\beta \vee \gamma))$	associativity of $\vee$
$\neg(\neg\alpha) \equiv \alpha$	double-negation elimination
$(\alpha \Rightarrow \beta) \equiv (\neg\beta \Rightarrow \neg\alpha)$	contraposition
$(\alpha \Rightarrow \beta) \equiv (\neg\alpha \vee \beta)$	implication elimination
$(\alpha \Leftrightarrow \beta) \equiv ((\alpha \Rightarrow \beta) \wedge (\beta \Rightarrow \alpha))$	biconditional elimination
$\neg(\alpha \wedge \beta) \equiv (\neg\alpha \vee \neg\beta)$	de Morgan
$\neg(\alpha \vee \beta) \equiv (\neg\alpha \wedge \neg\beta)$	de Morgan
$(\alpha \wedge (\beta \vee \gamma)) \equiv ((\alpha \wedge \beta) \vee (\alpha \wedge \gamma))$	distributivity of $\wedge$ over $\vee$
$(\alpha \vee (\beta \wedge \gamma)) \equiv ((\alpha \vee \beta) \wedge (\alpha \vee \gamma))$	distributivity of $\vee$ over $\wedge$

# Validity

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

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**:

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



# Satisfiability

A sentence is **satisfiable** (consistent) if it is true in **some** interpretation (i.e. it has a model)

e.g.,  $A \vee B, \quad C$

A sentence is **unsatisfiable** (inconsistent) if it is true in **no** interpretation (i.e. it has no model)

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

Satisfiability is connected to inference via the following:

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

i.e., prove  $\alpha$  by *reductio ad absurdum*

## Reasoning in propositional logic

Two decision problems:

- ◇ tautology checking ( $TAUT$ )
- ◇ satisfiability checking ( $SAT$ )

$TAUT$  and  $SAT$  are exponential in the size of the formula (the number of propositional symbols).

$SAT$  is NP-Complete and  $TAUT$  is Co-NP-complete.

# Reasoning methods

Reasoning methods can be (roughly) divided in two kinds:

## Model checking

- truth table enumeration (always exponential in  $n$ )

## Deduction

- generation of new sentences through the application of inference rules

## Efficient model checking

1. heuristic search, e.g., DPLL
2. local search in model space (sound but incomplete)  
e.g., min-conflicts-like, hill-climbing algorithms

Typically require translation of sentences into a **normal form**

## Conjunctive Normal Form

In **Conjunctive Normal Form (CNF)**:

- ◇ the KB is represented as a set of clauses.
- ◇ a *clause* is a disjunction of literals  $L_1 \vee L_2 \vee \dots \vee L_n$ .

### Example

$$\{\{A, \neg B \neg C\}, \{\neg A, B\}\} \Leftrightarrow (A \vee \neg B \vee \neg C) \wedge (\neg A \vee B)$$

- ◇ Every formula can be rewritten into an equivalent CNF (later).

## Search for a model

Propositional symbol = (boolean) variable

Model = boolean assignment

Formula = constraint to be satisfied

Heuristics DPLL: Davis–Putnam–Logemann–Loveland

- ◇ **early check** of satisfied or unsatisfiable formulae;
- ◇ **pure** symbols heuristics (always positive or negated);
- ◇ **unitary** formulae (only one literal);

# DPLL

**function** DPLL-SATISFIABLE?(*s*) **returns** *true* or *false*

**inputs:** *s*, a sentence in propositional logic

*clauses*  $\leftarrow$  the set of clauses in the CNF representation of *s*

*symbols*  $\leftarrow$  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*  $\leftarrow$  FIND-PURE-SYMBOL(*symbols*, *clauses*, *model*)

**if** *P* is non-null **then return** DPLL(*clauses*, *symbols*−*P*, [*P* = *value* | *model*])

*P*, *value*  $\leftarrow$  FIND-UNIT-CLAUSE(*clauses*, *model*)

**if** *P* is non-null **then return** DPLL(*clauses*, *symbols*−*P*, [*P* = *value* | *model*])

*P*  $\leftarrow$  FIRST(*symbols*); *rest*  $\leftarrow$  REST(*symbols*)

**return** DPLL(*clauses*, *rest*, [*P* = *true* | *model*]) **or**

DPLL(*clauses*, *rest*, [*P* = *false* | *model*])

## DPLL (implementation)

With a number of additional details this 1963 procedure can be made very efficient!

- separation of disjoint problems
- variable and value ordering
- intelligent backtracking (clause learning)
- random restarts
- clever indexing



## Local search: *GSAT*

Local search:

- ◇ start from a randomly chosen assignment
- ◇ compute best successor and continue search
- ◇ if fail then restart from new randomly chosen assignment

## Local search: *GSAT*

**function** GSAT(*formula max-restart max-search*) **returns** a truth assignment or failure

**for**  $i \leftarrow 1$  **to** *max-restart* **do**

$A \leftarrow$  a random truth assignment

**for**  $j \leftarrow 1$  **to** *max-search* **do**

**if**  $A$  satisfies *formula* **then return**  $A$

$A \leftarrow$  a random choice of one of the best successors of  $A$

**end**

**end**

**return** failure

Local search is efficient, but incomplete.

# WALKSAT

**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, around 0.5

*max-flips*, number of flips allowed before giving up

*model*  $\leftarrow$  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*  $\leftarrow$  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

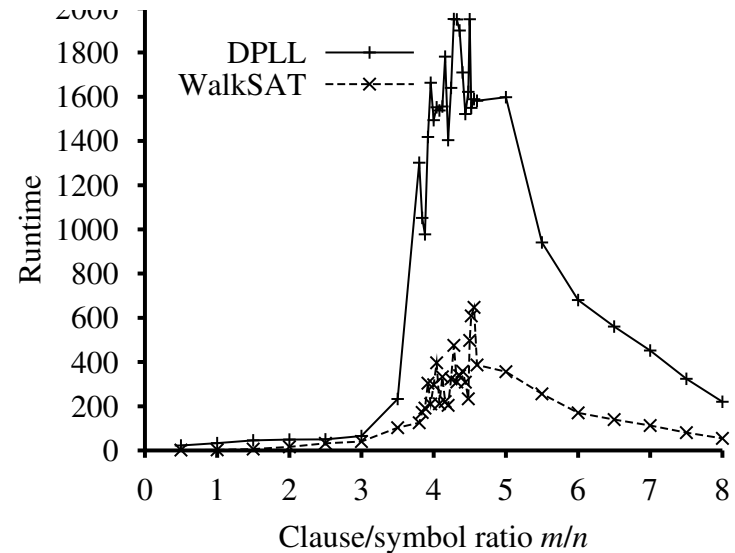
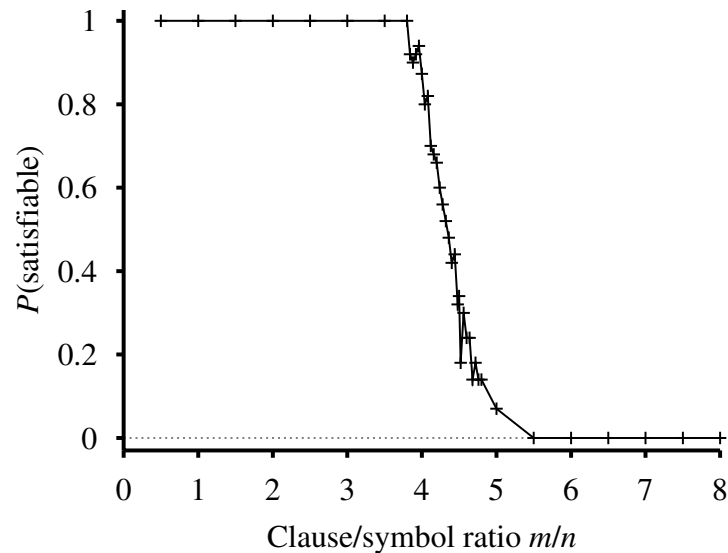
**return** *failure*

*Clauses* unsatisfiable ?

# Analysis of the search space

$m$  = number of clauses

$n$  = number of symbols (fixed to 50)



Difficult problems are around  $n/m = 4.3$

# Deduction in propositional logic

$$KB \vdash \alpha$$

Logical entailment can be computed by a syntactic process called **deduction** that manipulates (propositional) sentences.

Deduction can be characterized as a search:

- ◇ Init state is the formula representing the KB,
- ◇ the operators are the **inference rules**
- ◇ the final state is the formula to be proven

## Inference Rules

◇ The inference rules  $\mathcal{R}$  are typically written:

$$\frac{A_1 \cdots A_n}{A}$$

$$\frac{\textit{premises}}{\textit{conclusions}}$$

◇ Some deductive systems include also Axioms  $Ax$ , but they can be replaced by inference rules:

$$\frac{}{A}$$

for each  $A \in Ax$

## Theorems

◇ Let  $\Gamma$  a set of formulae.  $A$  is derived from  $\Gamma$  ( $\Gamma \vdash A$ ) if there exists a sequence of formulae  $A_1, \dots, A_n$  such that:

- $A$  is  $A_n$
- for every  $i$  between 1 and  $n$ , either  $A_i \in \Gamma$  or  $A_i$  is a **direct derivation** of the formulae in the previous sequence.

The sequence  $A_1, \dots, A_n$  is a *proof* of  $A$  from  $\Gamma$ .

The elements of  $\Gamma$  are called *premises*, or *hypotheses*, or else *assumptions* of  $A$ .

## Basic properties

◇ A deduction method  $\mathcal{R}$  is *sound* if for every formula  $A$ ,

$$\vdash_{\mathcal{R}} A \text{ implies } \models A$$

◇ A deduction method  $\mathcal{R}$  is *complete* wrt a set of formulae  $\Gamma$ , if for every  $A \in \Gamma$ ,

$$\models A \text{ implies } \vdash_{\mathcal{R}} A$$

◇ Soundness and completeness:

$$\vdash \equiv \models$$



# Deduction

$\Gamma \models A$  is the basic reasoning problem

- direct proof:

$$\Gamma \vdash_{\mathcal{R}} A$$

- proof by refutation (reductio ad absurdum):

$\Gamma \cup \{\neg A\}$  is unsatisfiable

from  $\Gamma \cup \{\neg A\}$  we get inconsistency

# Inference rules for propositional logic

## Modus Ponens

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

## And Elimination

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

## Examples

$$\frac{man \Rightarrow mortal, \quad man}{mortal}$$

$$\Gamma = \{feline \Rightarrow animal, cat \Rightarrow feline, cat\}$$

$$\Gamma \vdash_{MP} animal$$

## Deduction in propositional logic

- **hilbert system** (MP + axiom schemata): first, intuitive, not mechanizable;
- **tableau**: intuitive and mechanizable;
- **resolution**: born for automated deduction ... basis of PROLOG language

Forward/backward reasoning:

sound and complete (and polynomial) only for Horn clauses

# Horn Clauses

Horn Form (restricted)

KB = *conjunction* of *Horn clauses*

Horn clause =

- ◇ proposition symbol; or
- ◇ (conjunction of symbols)  $\Rightarrow$  symbol

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

Definite Clauses exactly one positive literal

## Forward and backward chaining

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

$$\frac{\alpha_1, \dots, \alpha_n, \quad \alpha_1 \wedge \dots \wedge \alpha_n \Rightarrow \beta}{\beta}$$

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 (e.g.  $Q$ )

$$P \Rightarrow Q$$

$$L \wedge M \Rightarrow P$$

$$B \wedge L \Rightarrow M$$

$$A \wedge P \Rightarrow L$$

$$A \wedge B \Rightarrow L$$

$A$

$B$

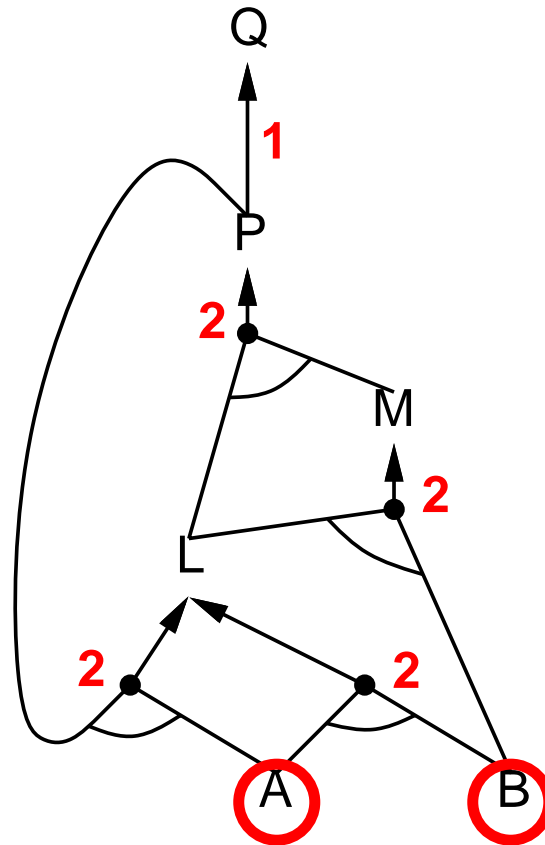
# Forward chaining algorithm

```
function PL-FC-ENTAILS?(KB, q) returns true or false
  inputs: KB, a set of propositional Horn clauses
           q, the query, a proposition symbol
  local variables: count, a table indexed by clause, init # premises
                    inferred, a table indexed by symbol, init false
                    agenda, a list of symbols, init 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)

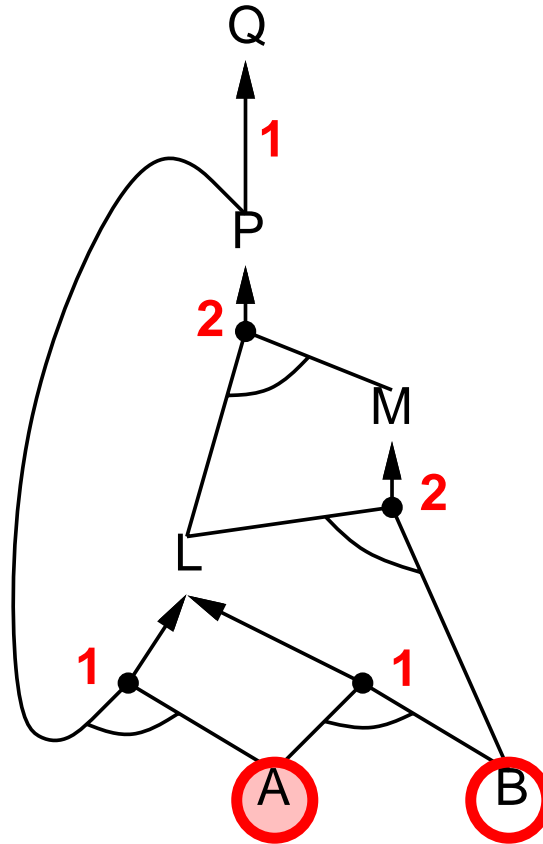
  return false
```

# Forward chaining example

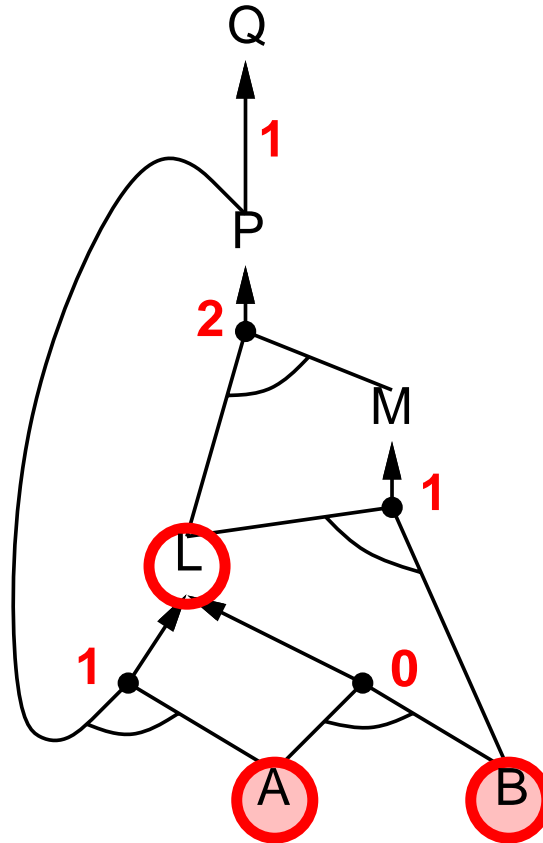




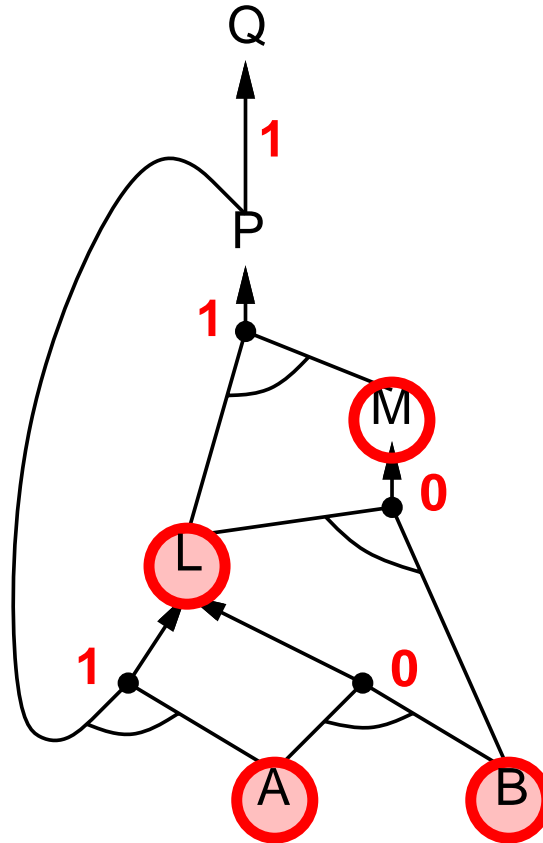
# Forward chaining example



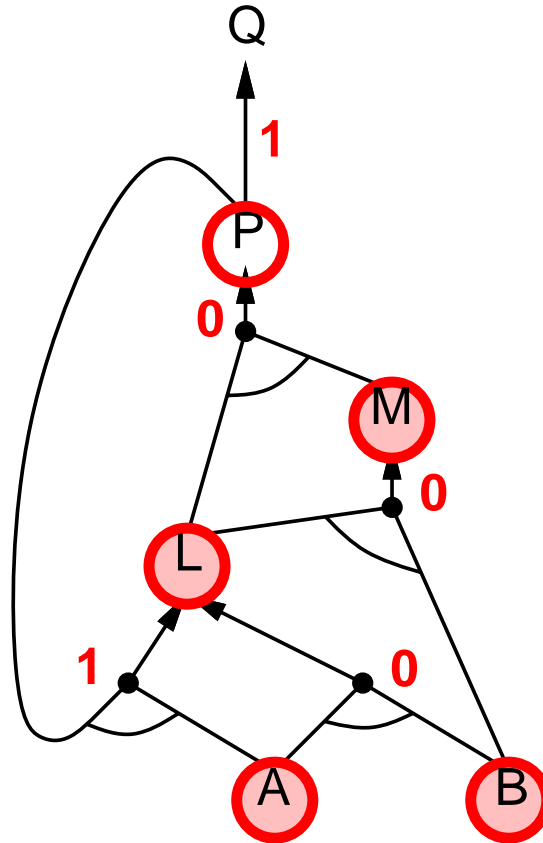
# Forward chaining example



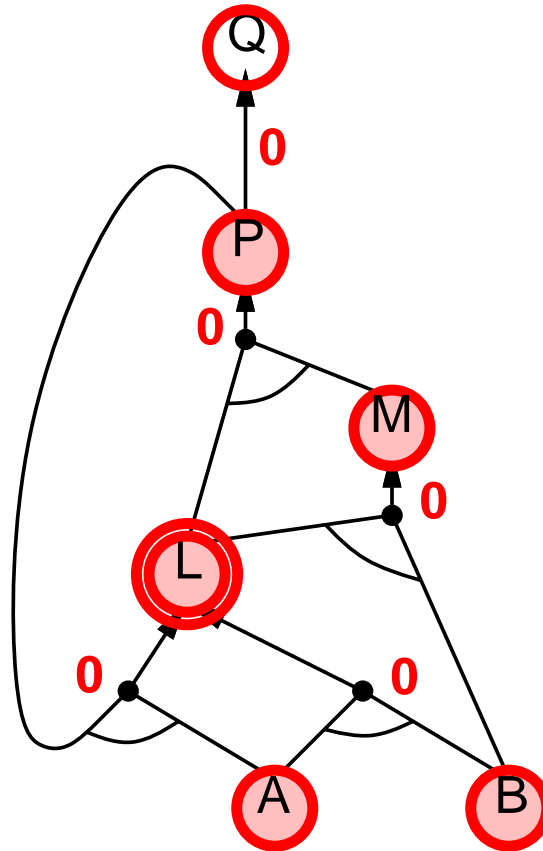
# Forward chaining example



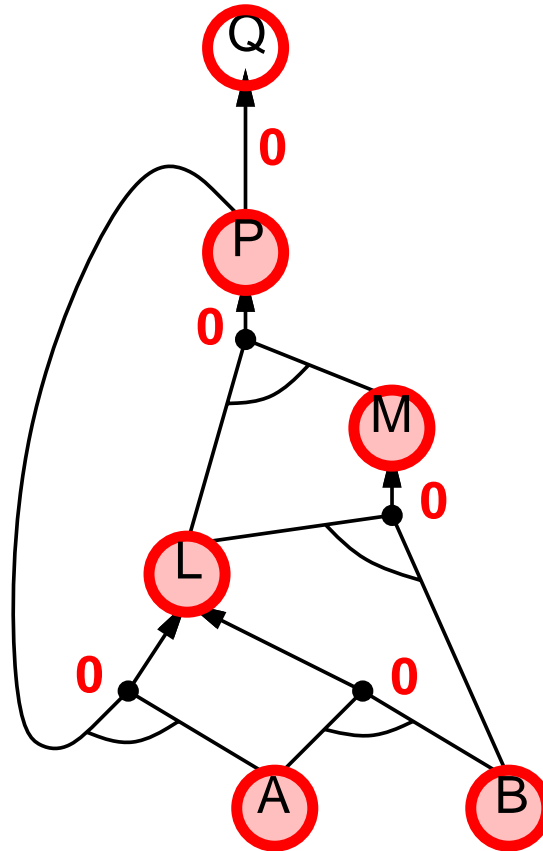
# Forward chaining example



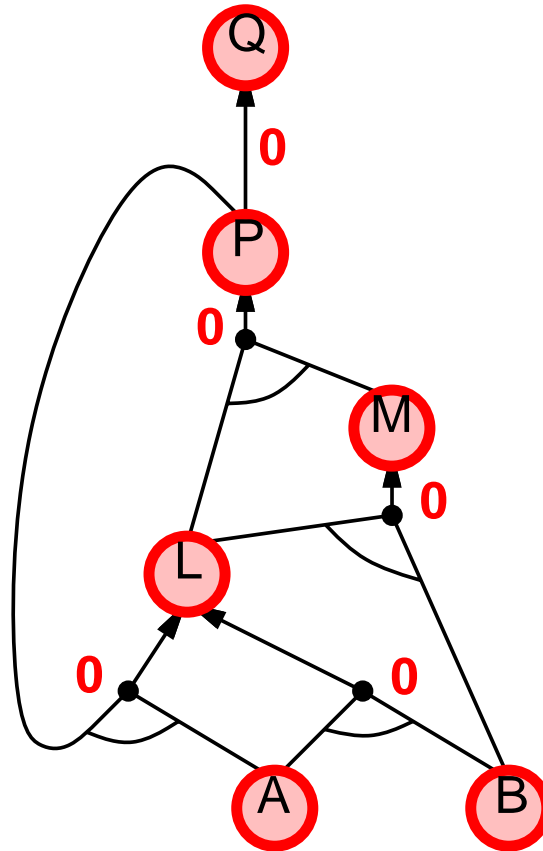
# Forward chaining example



# Forward chaining example



# Forward chaining example



## Proof of completeness

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

1. FC 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  $m$

**Proof:** Suppose a clause  $a_1 \wedge \dots \wedge a_k \Rightarrow b$  is false in  $m$

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

Therefore the algorithm has not 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  $m$



## 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  $q$

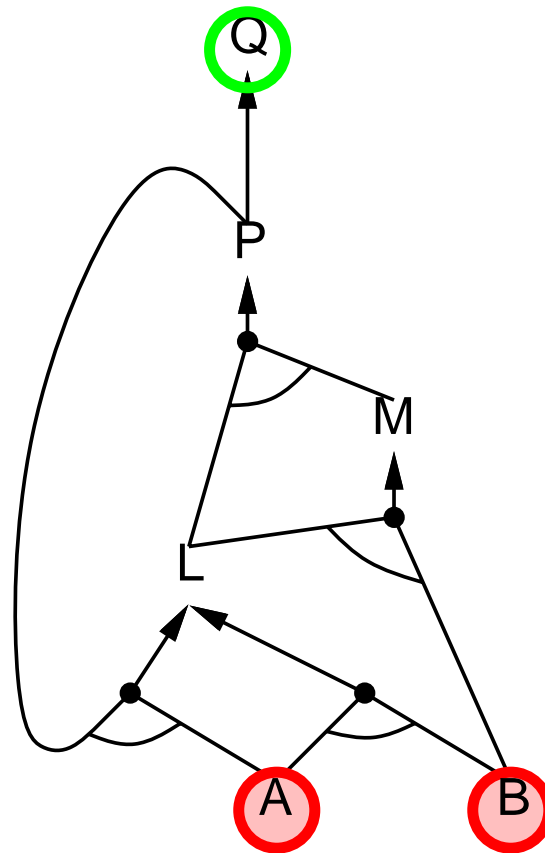
Avoid loops: check if new subgoal is already on the goal stack

Avoid repeated work: check if new subgoal

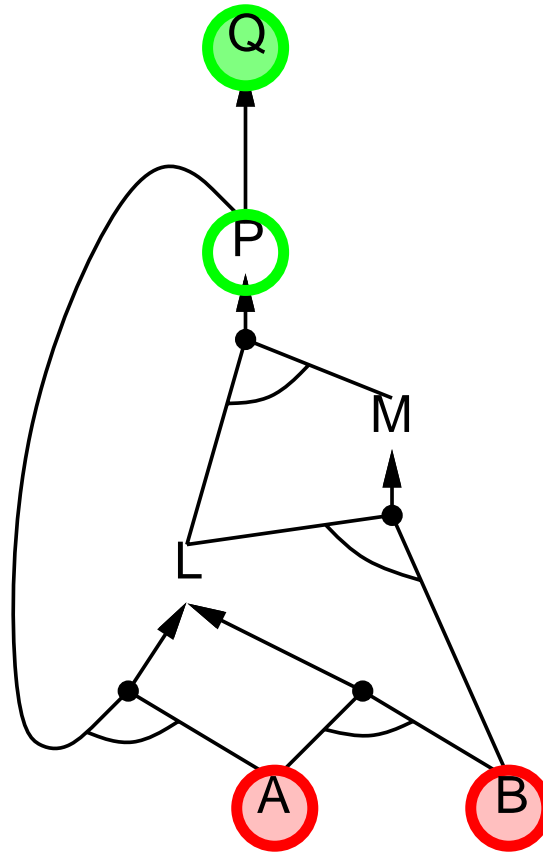
1) has already been proved true, or

2) has already failed

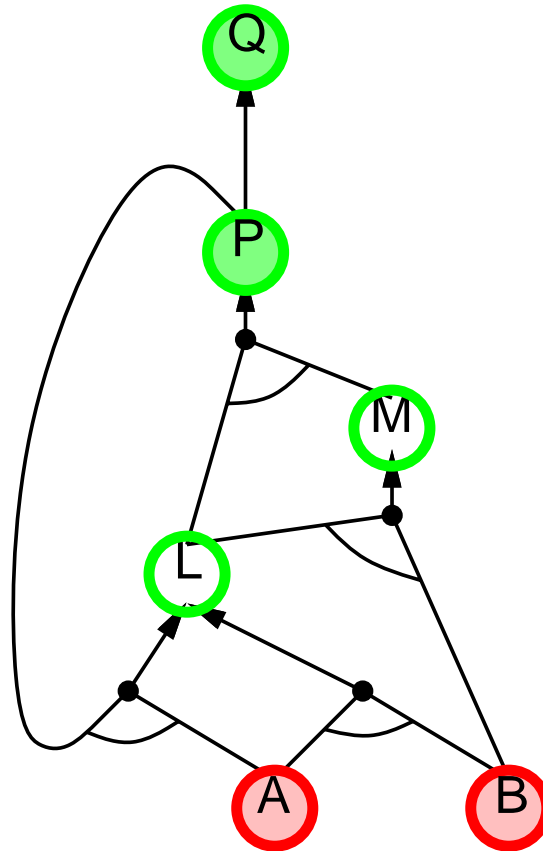
# Backward chaining example



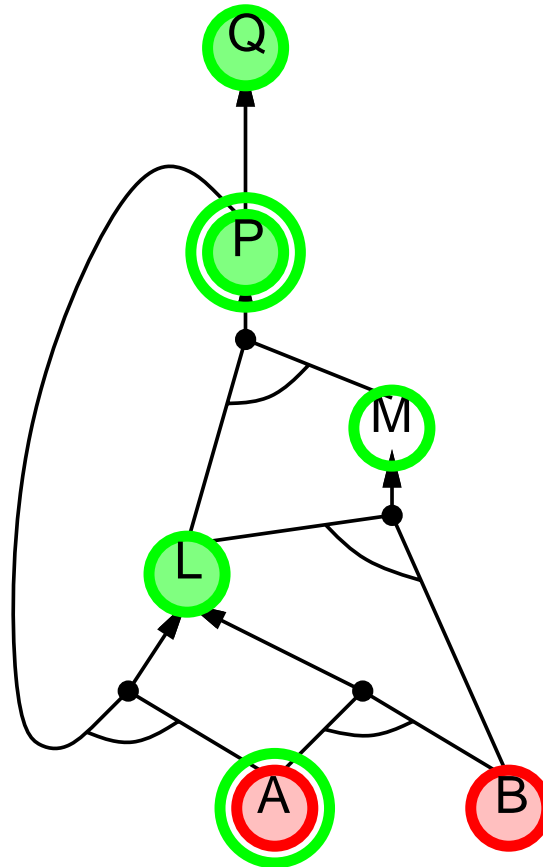
# Backward chaining example



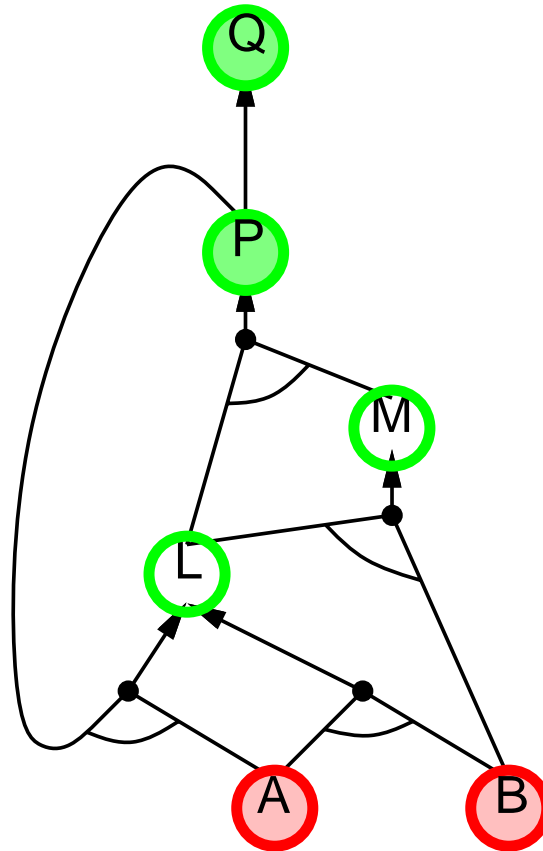
# Backward chaining example



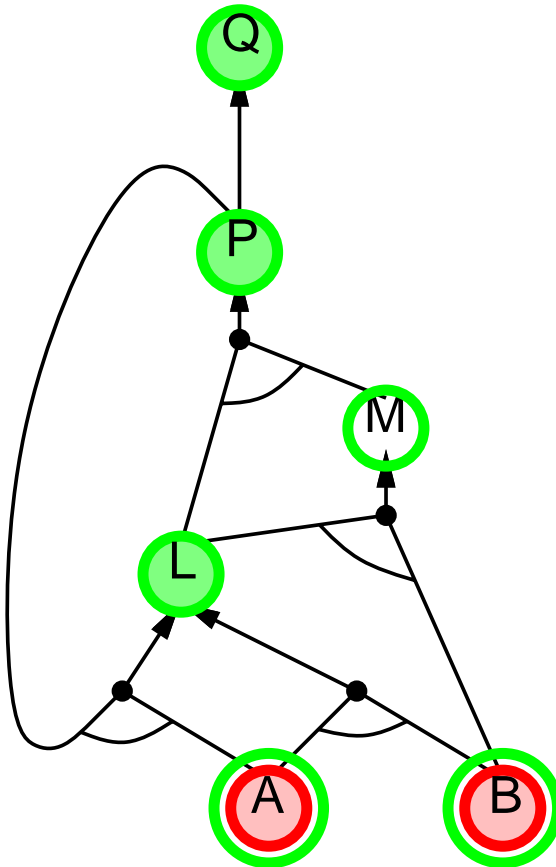
# Backward chaining example



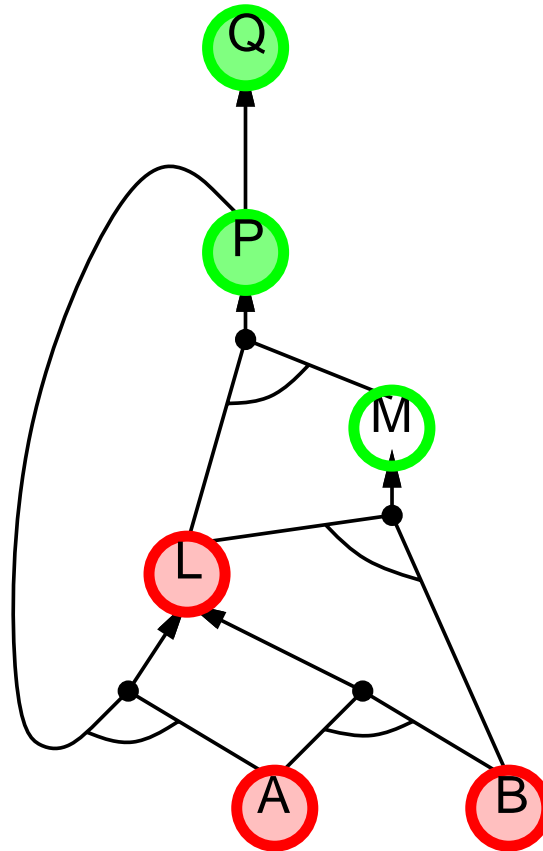
# Backward chaining example



# Backward chaining example

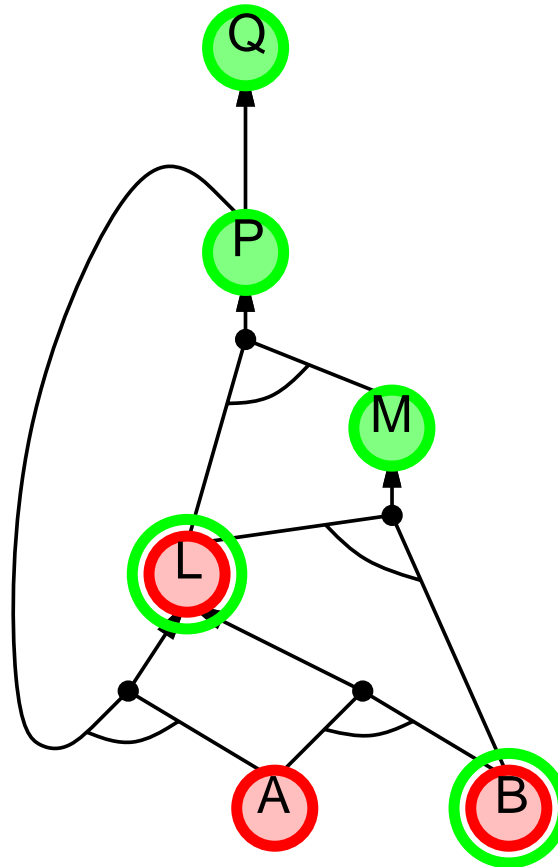


# Backward chaining example

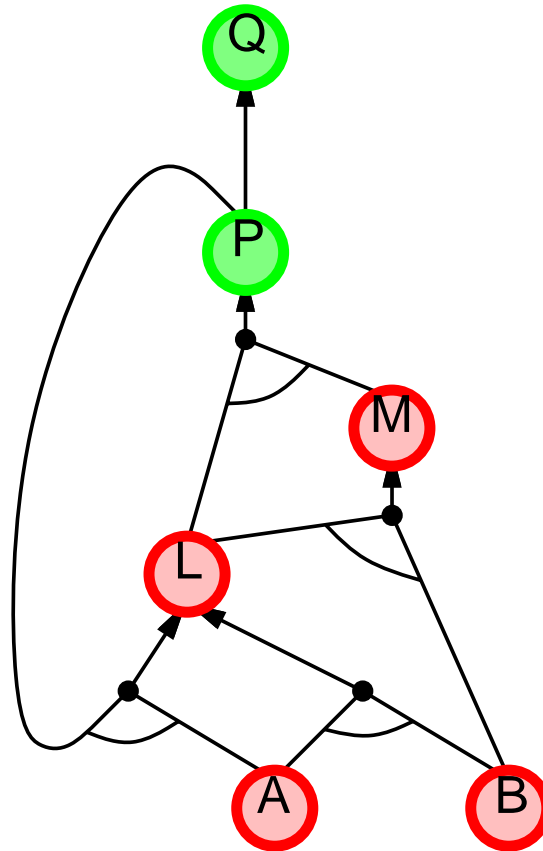




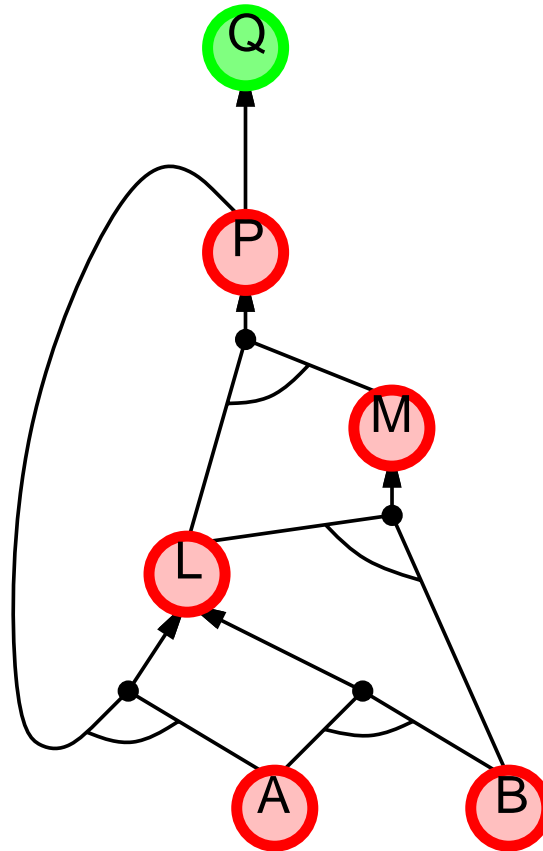
# Backward chaining example



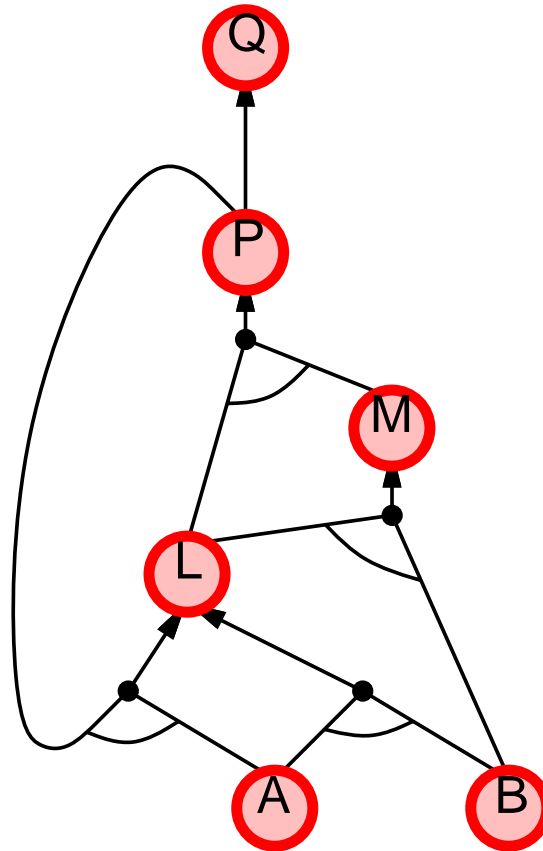
# Backward chaining example



# Backward chaining example



# Backward chaining example



## 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

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

## Summary

- ◇ syntax and semantics of propositional logic
- ◇ model checking
- ◇ inference
- ◇ efficient model checking
- ◇ efficient inference (with Horn clauses)