

# CS 380: Artificial Intelligence

---

## Lecture 9: Propositional Logic

---

# Recall...

---

- Logical Agents:
  - can reason from the knowledge they have
  - can make deductions from their perceptions, and adjust their behavior accordingly
- Knowledge: set of sentences in a formal language
- Logic:
  - A formal language for representing information
  - Syntax: defines sentences in the language
  - Semantics: defines the “meaning” of sentences
  - Entailment: defines which sentences can be derived from others

## Entailment

Entailment means that one thing **follows from** another:

$$KB \models \alpha$$

Knowledge base  $KB$  entails sentence  $\alpha$

if and only if

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

E.g., the KB containing “the Giants won” and “the Reds won”  
entails “Either the Giants won or the Reds won”

E.g.,  $x + y = 4$  entails  $4 = x + y$

Entailment is a relationship between sentences (i.e., **syntax**)  
that is based on **semantics**

Note: brains process **syntax** (of some sort)

# Models

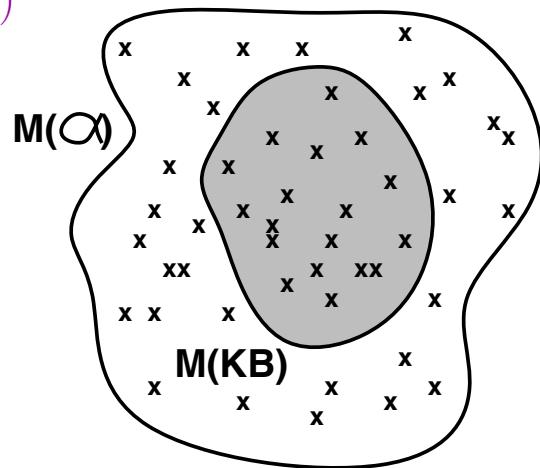
Logicians typically think in terms of **models**, which are formally structured worlds with respect to which truth can be evaluated

We say  $m$  is a model of a sentence  $\alpha$  if  $\alpha$  is true in  $m$

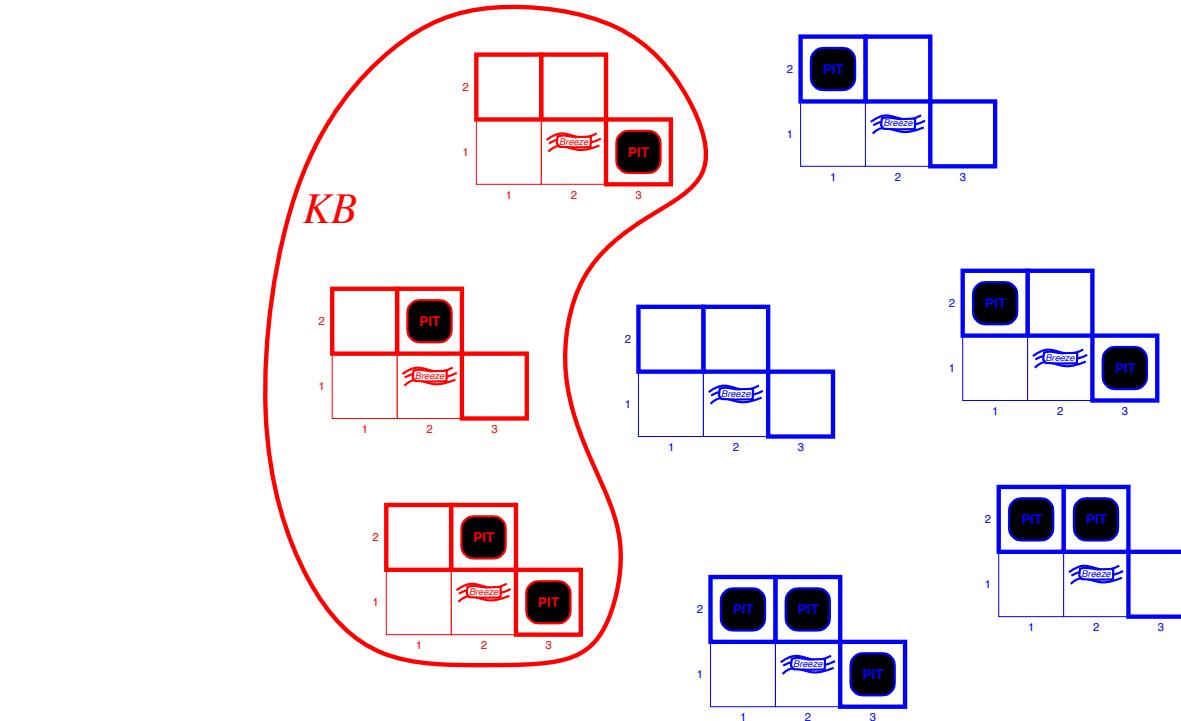
$M(\alpha)$  is the set of all models of  $\alpha$

Then  $KB \models \alpha$  if and only if  $M(KB) \subseteq M(\alpha)$

E.g.  $KB =$  Giants won and Reds won  
 $\alpha =$  Giants won

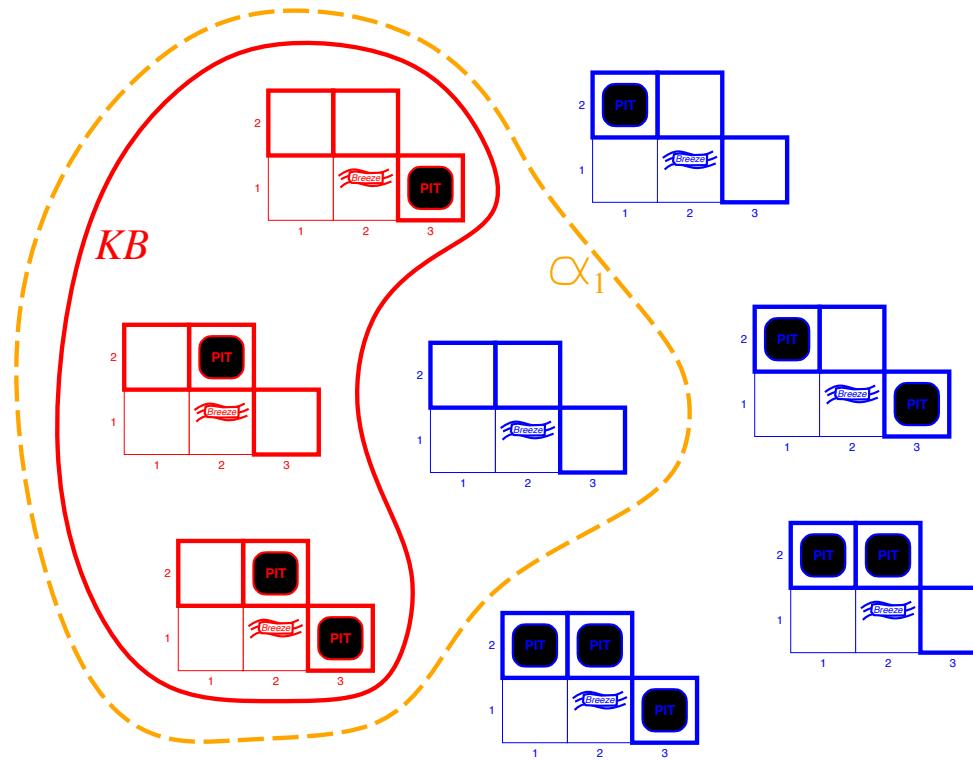


# Wumpus models



*KB* = wumpus-world rules + observations

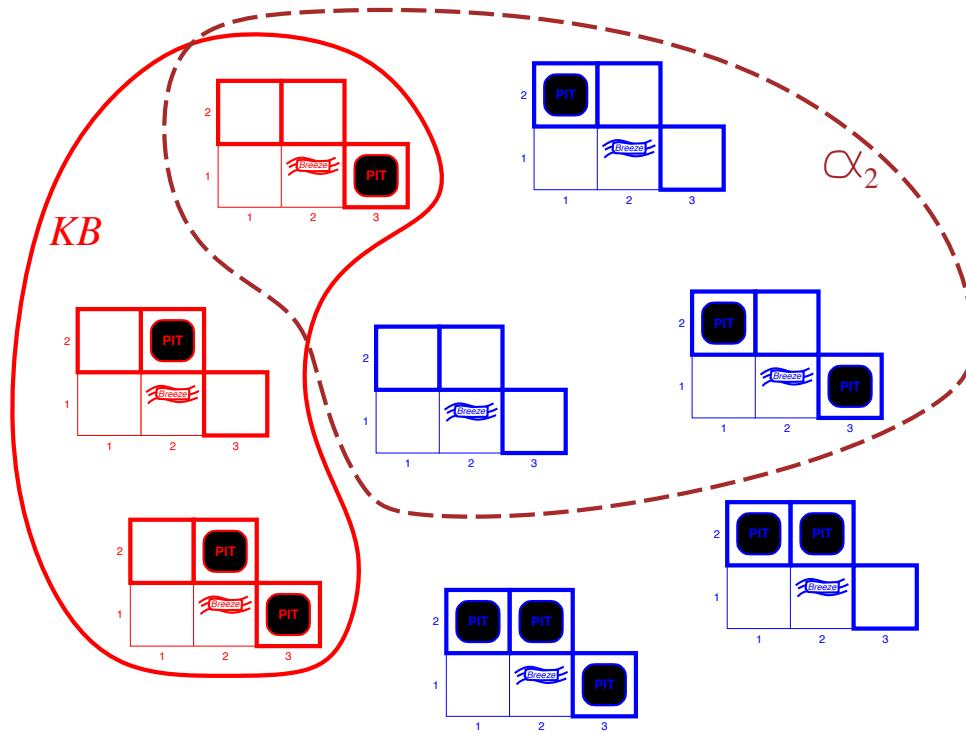
# Wumpus models



$KB = \text{wumpus-world rules} + \text{observations}$

$\alpha_1 = "[1,2] \text{ is safe}", KB \models \alpha_1$ , proved by model checking

# Wumpus models



$KB$  = wumpus-world rules + observations

$\alpha_2$  = “[2,2] is safe”,  $KB \not\models \alpha_2$

# Entailment & Inference

---

- Many different logics can be defined:
  - Propositional Logic
  - First-order Logic
  - Description Logics (subset of FOL)
  - Etc.
- Entailment:
  - Can be determined via semantics (co-NP-complete)
- Inference
  - Procedure to determine entailment

## Inference

$KB \vdash_i \alpha$  = sentence  $\alpha$  can be derived from  $KB$  by procedure  $i$

Consequences of  $KB$  are a haystack;  $\alpha$  is a needle.

Entailment = needle in haystack; inference = finding it

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  $KB$ .

## Propositional logic: Syntax

Propositional logic is the simplest logic—illustrates basic ideas

The proposition symbols  $P_1, P_2$  etc are sentences

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: Semantics

Each model specifies true/false for each proposition symbol

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

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

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 <b>and</b>	$S_2$ is true
$S_1 \vee S_2$	is true iff	$S_1$	is true <b>or</b>	$S_2$ is true
$S_1 \Rightarrow S_2$	is true iff	$S_1$	is false <b>or</b>	$S_2$ is true
i.e.,	is false iff	$S_1$	is true <b>and</b>	$S_2$ is false
$S_1 \Leftrightarrow S_2$	is true iff	$S_1 \Rightarrow S_2$	is true <b>and</b>	$S_2 \Rightarrow S_1$ is true

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

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

## 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 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  $[i, j]$ .

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

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

$$B_{2,1}$$

“Pits cause breezes in adjacent squares”

## 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  $[i, j]$ .

### Rules

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

$$R_2 \quad \neg B_{1,1}$$

$$R_3 \quad B_{2,1}$$

“Pits cause breezes in adjacent squares”

$$R_4 \quad B_{1,1} \Leftrightarrow (P_{1,2} \vee P_{2,1})$$

$$R_5 \quad B_{2,1} \Leftrightarrow (P_{1,1} \vee P_{2,2} \vee P_{3,1})$$

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

## Truth tables for inference

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

Enumerate rows (different assignments to symbols),  
if  $KB$  is true in row, check that  $\alpha$  is too

## Truth tables for inference

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

Enumerate rows (different assignments to symbols),  
if  $KB$  is true in row, check that  $\alpha$  is too

## Inference by enumeration

Depth-first enumeration of all models is sound and complete

**function** TT-ENTAILS?(*KB,  $\alpha$* ) **returns** *true* or *false*

**inputs:** *KB*, the knowledge base, a sentence in propositional logic  
     $\alpha$ , the query, a sentence in propositional logic

*symbols*  $\leftarrow$  a list of the proposition symbols in *KB* and  $\alpha$

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

---

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

**if** EMPTY?(*symbols*) **then**

**if** PL-TRUE?(*KB, model*) **then return** PL-TRUE?( $\alpha, model$ )  
    **else return** *true*

**else do**

$P \leftarrow \text{FIRST}(symbols); rest \leftarrow \text{REST}(symbols)$

**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:

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

$(\alpha \wedge \beta) \equiv (\beta \wedge \alpha)$	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 and satisfiability

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

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

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

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

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.,  $A \wedge \neg A$

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*

# Proof methods

Proof methods divide into (roughly) two kinds:

## 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

- 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 =

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

E.g.,  $C \wedge D \Rightarrow B$

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

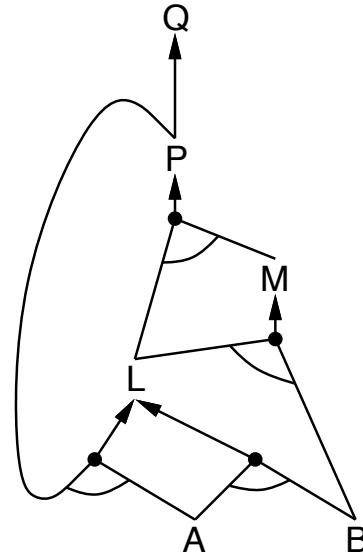
$$\frac{\alpha_1, \dots, \alpha_n, \quad \alpha_1 \wedge \dots \wedge \alpha_n \Rightarrow \beta}{\beta}$$

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

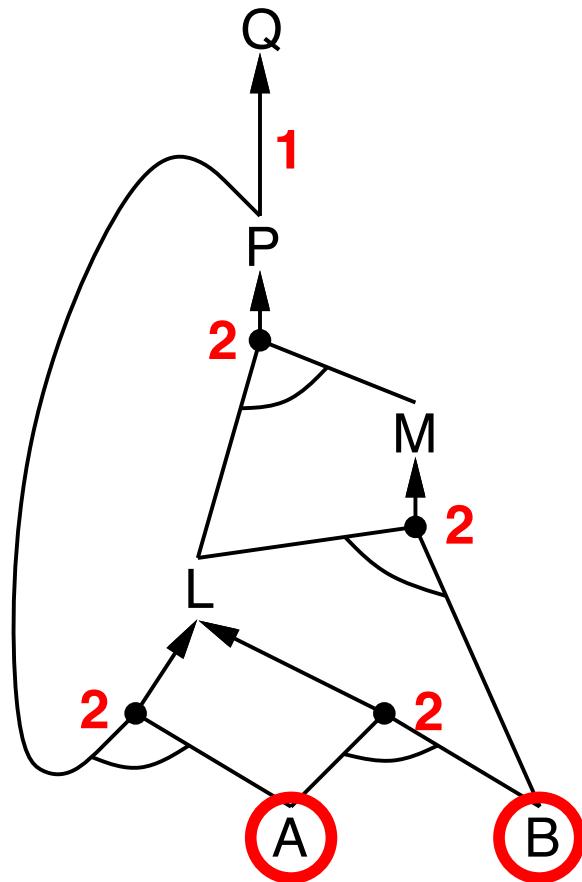
$$\begin{aligned} P \Rightarrow Q \\ L \wedge M \Rightarrow P \\ B \wedge L \Rightarrow M \\ A \wedge P \Rightarrow L \\ A \wedge B \Rightarrow L \\ A \\ B \end{aligned}$$


## Forward chaining algorithm

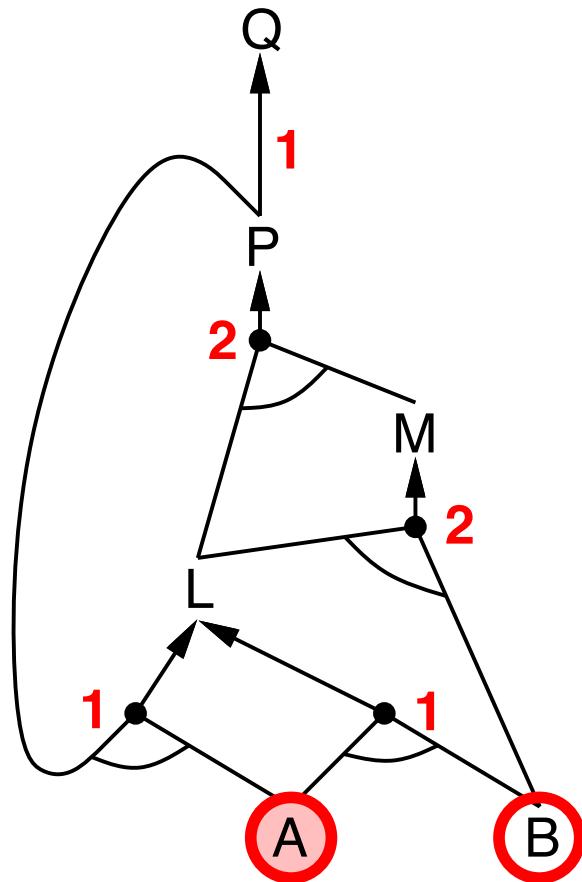
```
function PL-FC-ENTAILS?(KB, q) returns true or false
    inputs: KB, the knowledge base, a set of propositional Horn clauses
            q, the query, a proposition symbol
    local variables: count, a table, indexed by clause, initially the number of premises
                    inferred, a table,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

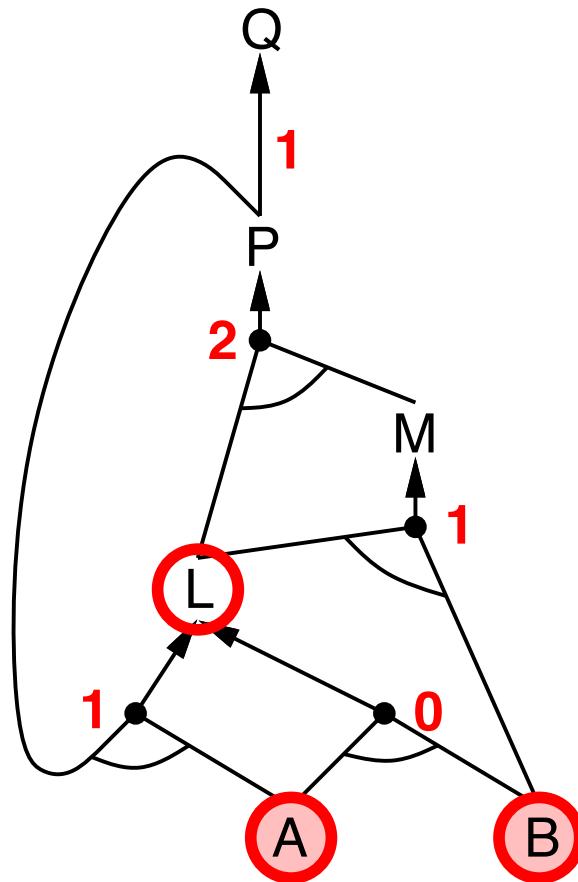
## Forward chaining example



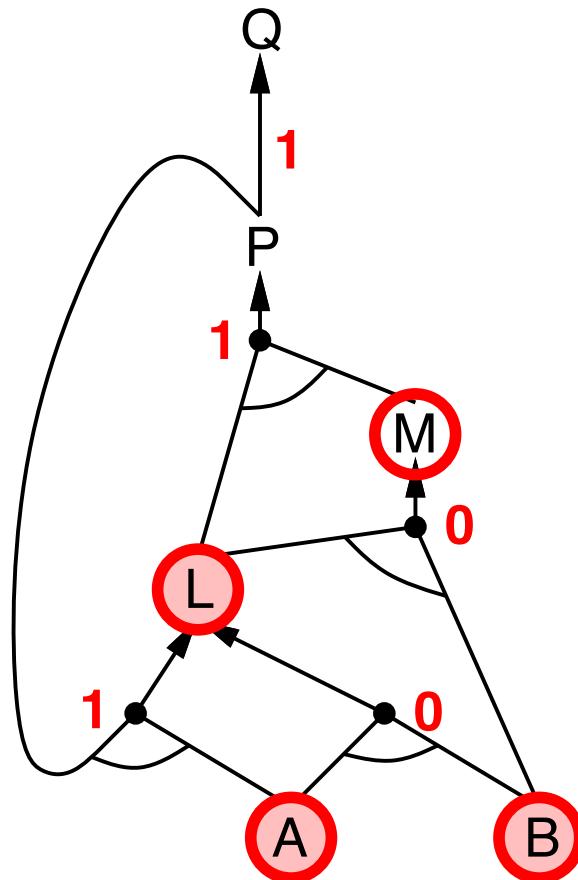
## Forward chaining example



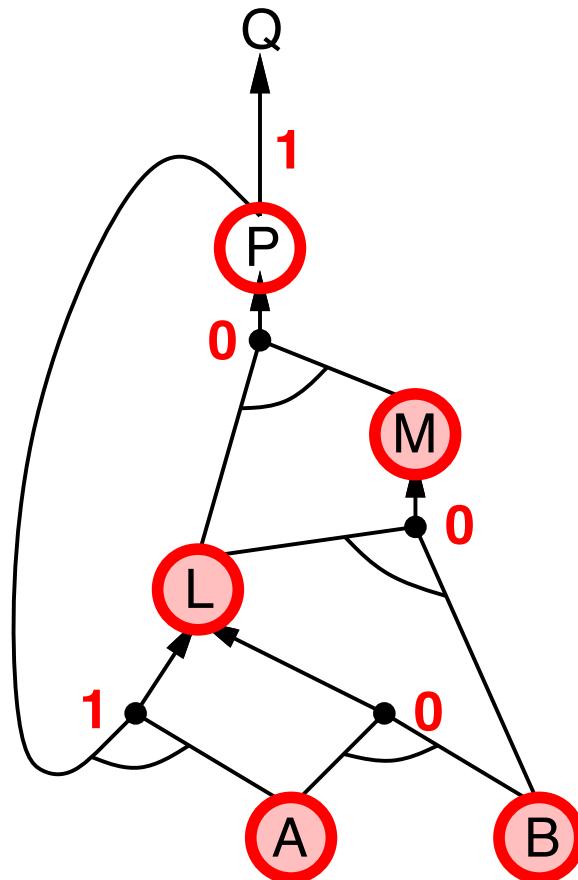
## Forward chaining example



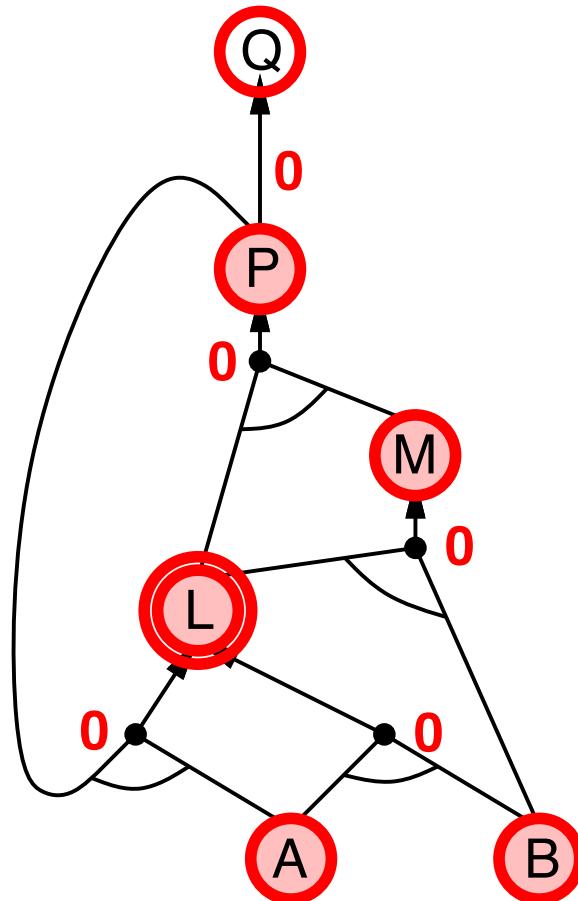
## Forward chaining example



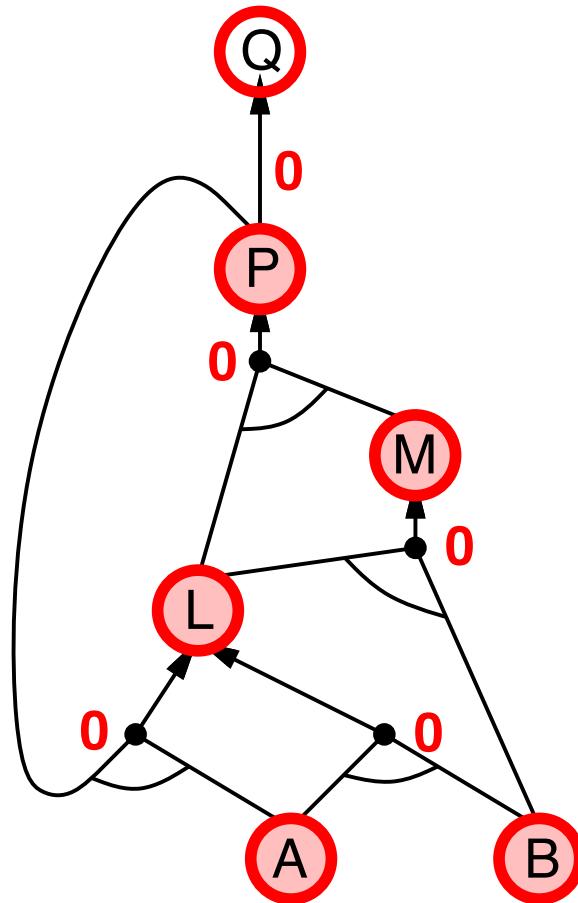
## Forward chaining example



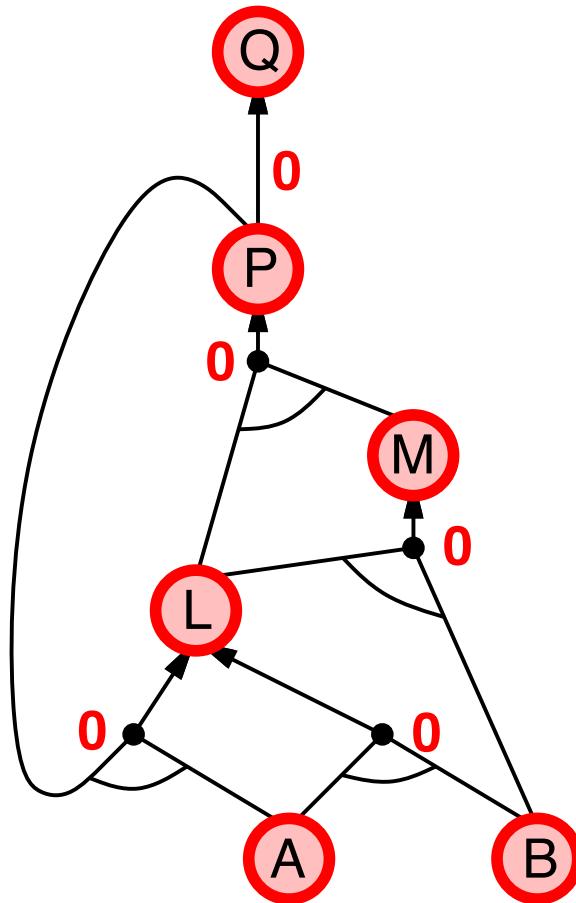
## Forward chaining example



## 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  $m$

**General idea:** construct any model of  $KB$  by sound inference, check  $\alpha$

## Backward chaining

Idea: work backwards from the query  $q$ :

to prove  $q$  by BC,

check if  $q$  is known already, or

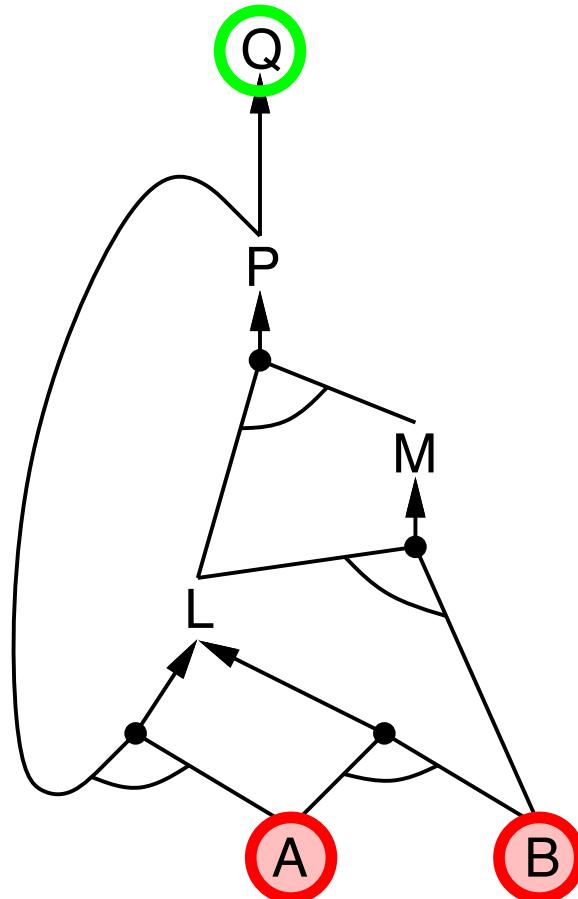
prove by BC all premises of some rule concluding  $q$

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

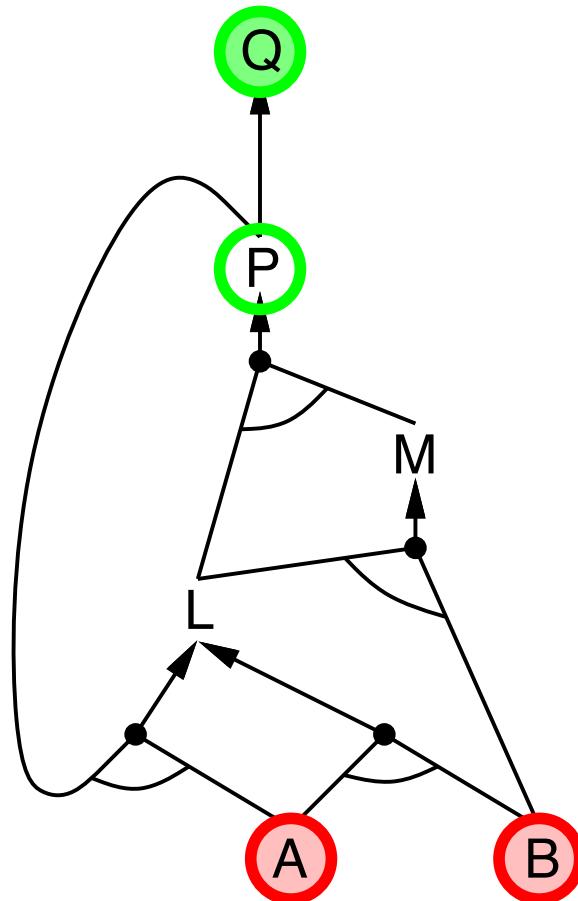
Avoid repeated work: check if new subgoal

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

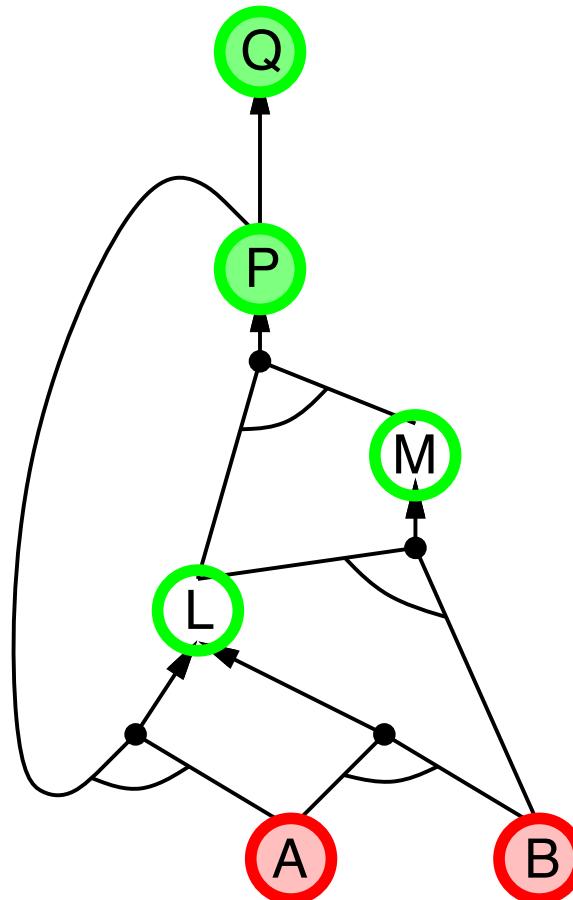
## Backward chaining example



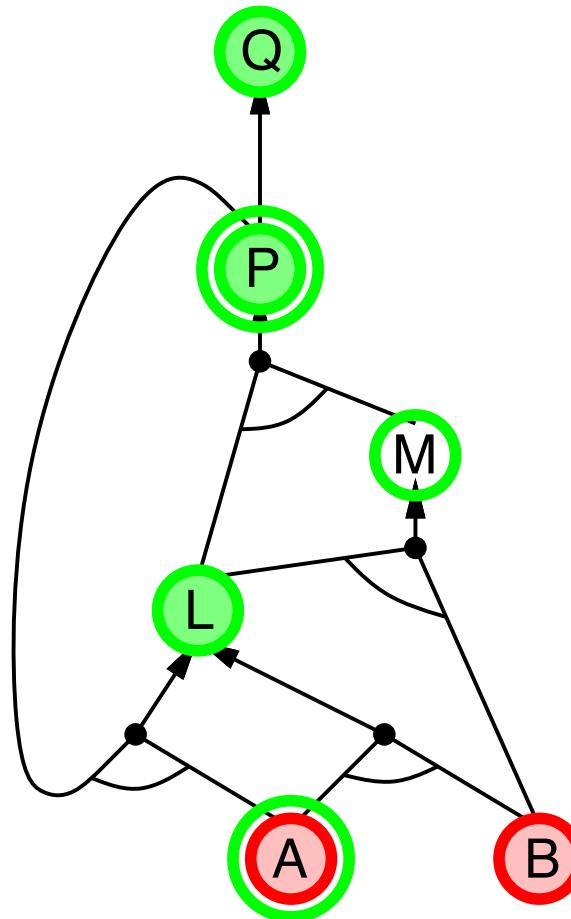
## Backward chaining example



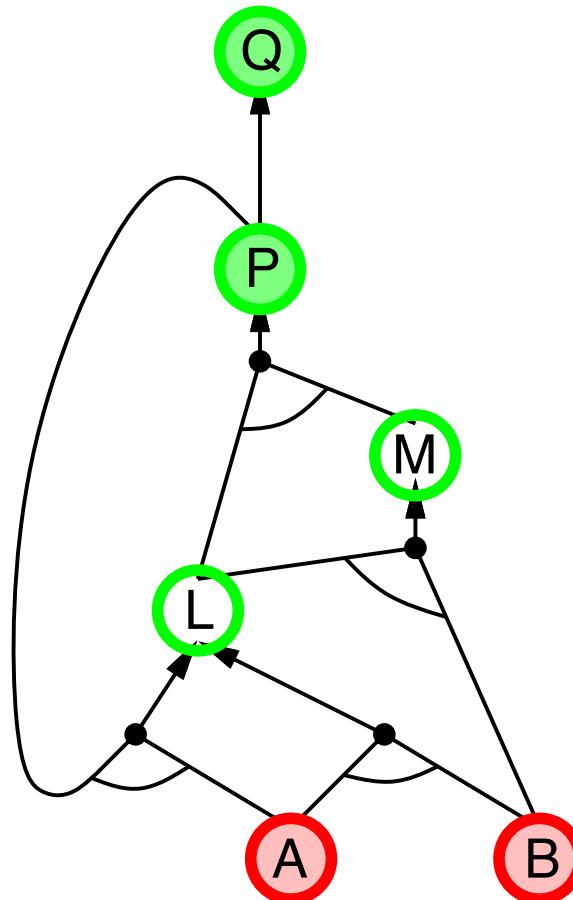
## Backward chaining example



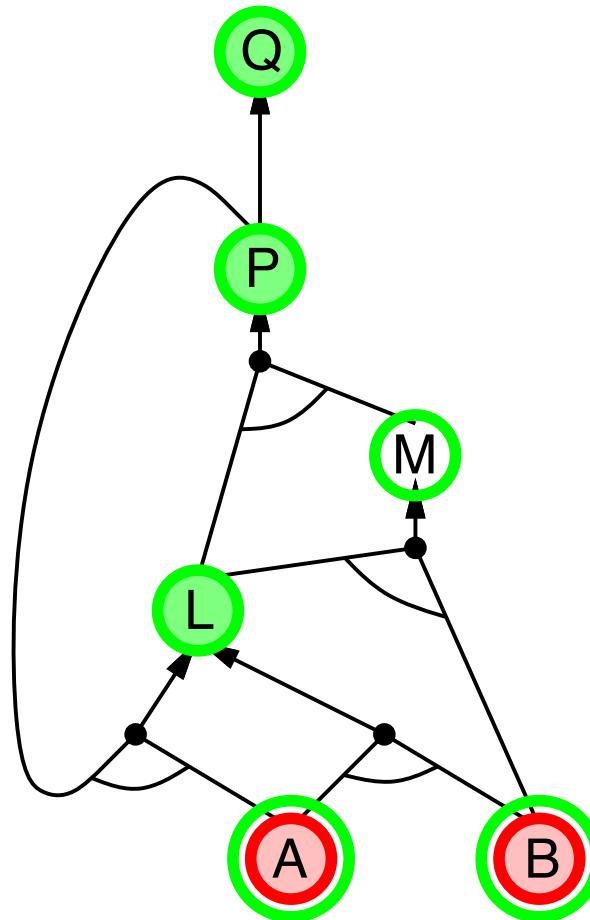
## Backward chaining example



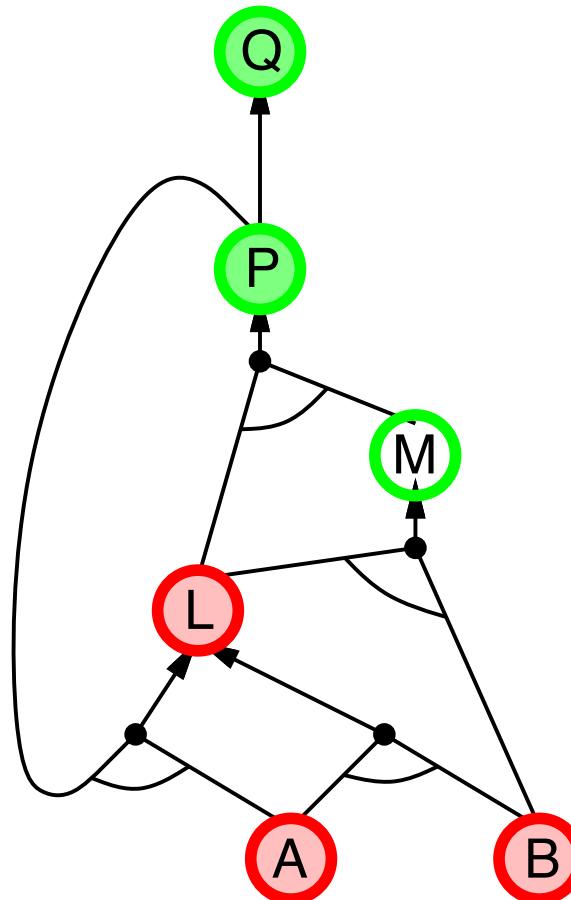
## Backward chaining example



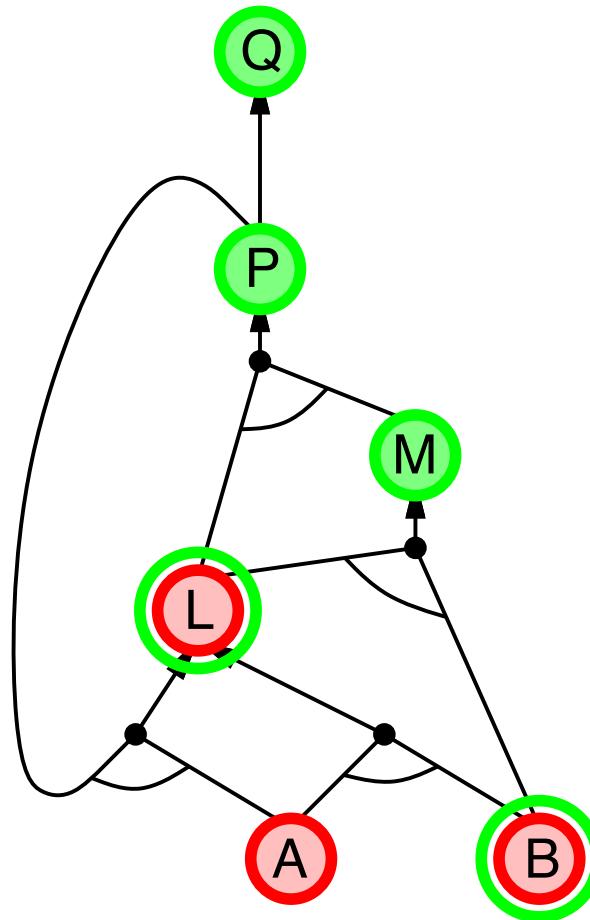
## Backward chaining example



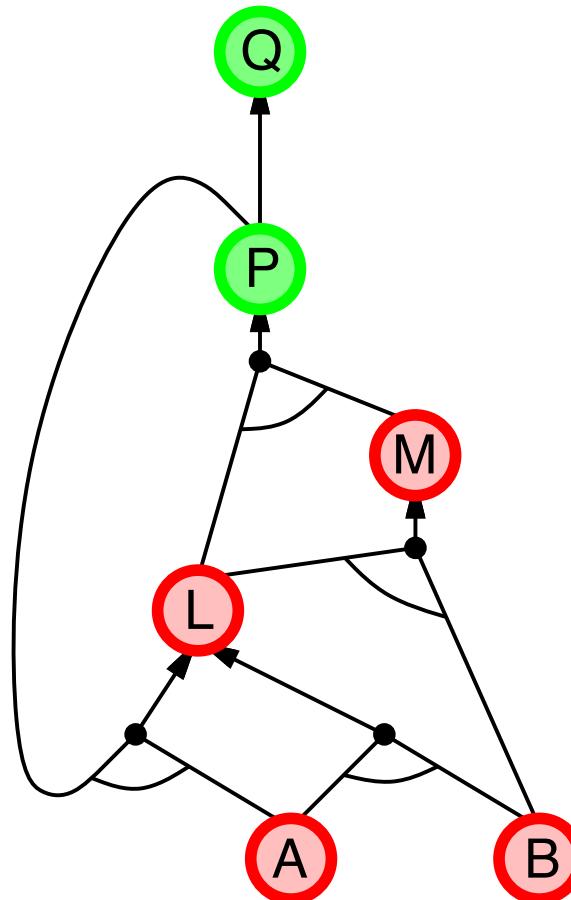
## Backward chaining example



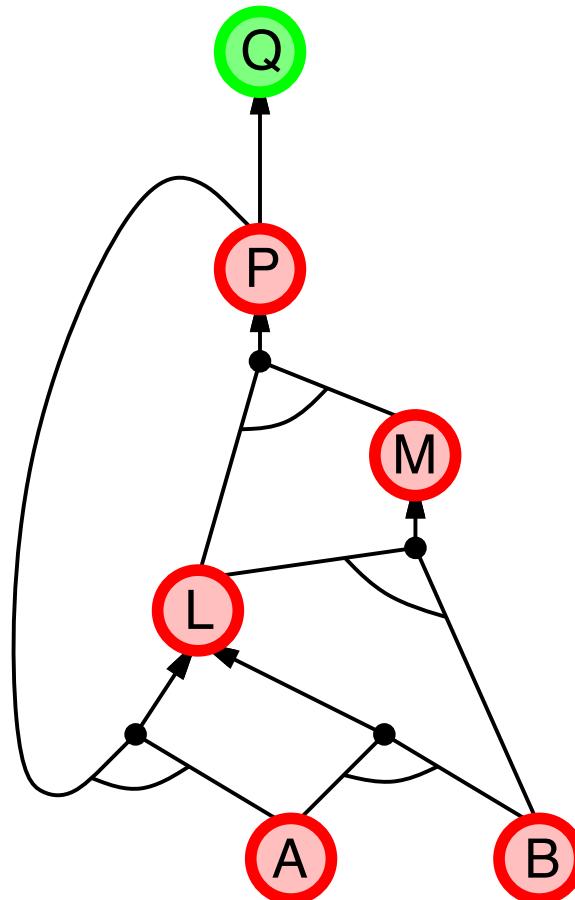
## Backward chaining example



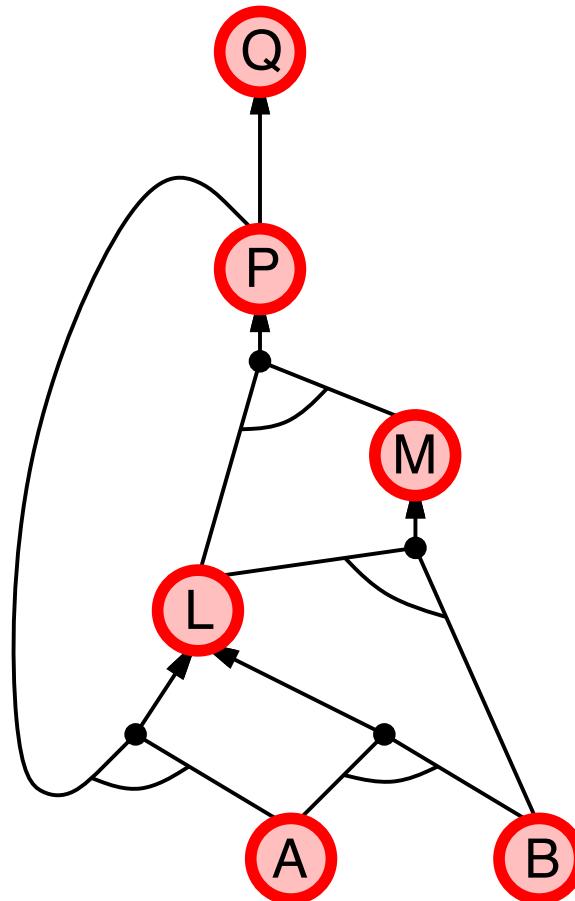
## Backward chaining example



## 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

# Resolution

Conjunctive Normal Form (CNF—universal)

**conjunction of disjunctions of literals**  
**clauses**

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

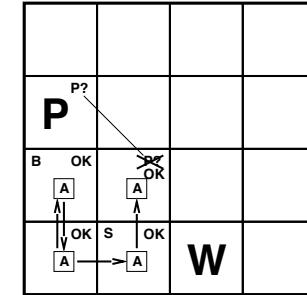
Resolution inference rule (for CNF): complete for propositional logic

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

where  $\ell_i$  and  $m_j$  are complementary literals. E.g.,

$$\frac{P_{1,3} \vee P_{2,2}, \quad \neg P_{2,2}}{P_{1,3}}$$

Resolution is sound and complete for propositional logic



## Conversion to CNF

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

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 ( $\vee$  over  $\wedge$ ) and flatten:

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

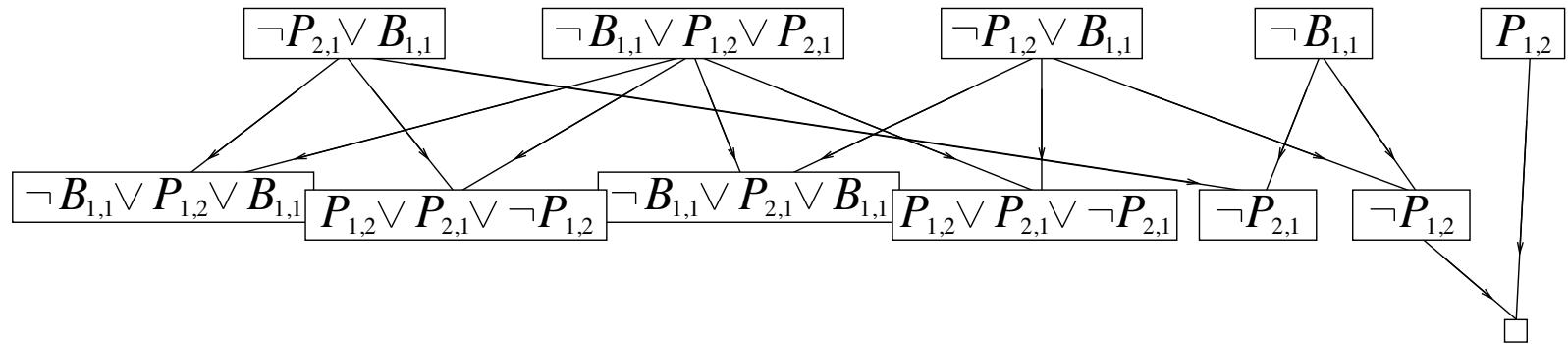
# Resolution algorithm

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

```
function PL-RESOLUTION( $KB, \alpha$ ) returns true or false
    inputs:  $KB$ , the knowledge base, a sentence in propositional logic
             $\alpha$ , the query, a sentence in propositional logic
     $clauses \leftarrow$  the set of clauses in the CNF representation of  $KB \wedge \neg\alpha$ 
     $new \leftarrow \{ \}$ 
    loop do
        for each  $C_i, C_j$  in  $clauses$  do
             $resolvents \leftarrow$  PL-RESOLVE( $C_i, C_j$ )
            if  $resolvents$  contains the empty clause then return true
             $new \leftarrow new \cup resolvents$ 
        if  $new \subseteq clauses$  then return false
         $clauses \leftarrow clauses \cup new$ 
```

# Resolution example

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



# 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