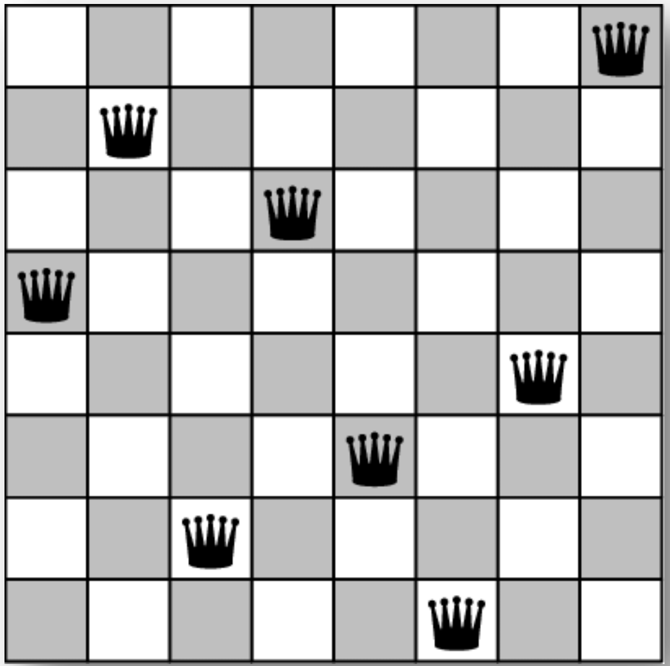
Introduction to Artificial Intelligence

Roman Barták
Department of Theoretical Computer Science and Mathematical Logic

Constraint satisfaction – just to recall

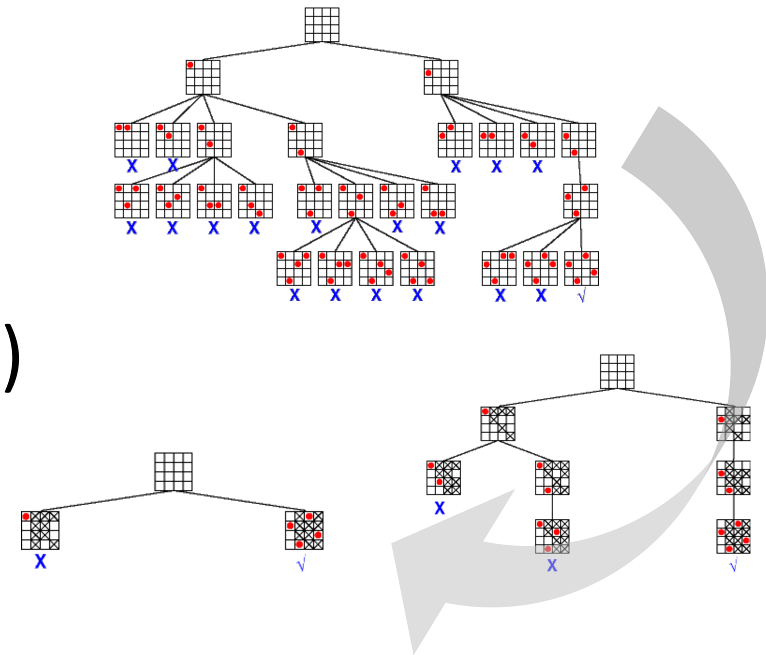


A problem-solving approach for combinatorial problems formulated as **constraint satisfaction problems**:

- construct a model (variables, domains, constraints)
- use a general constraint solver

Constraint satisfaction combines:

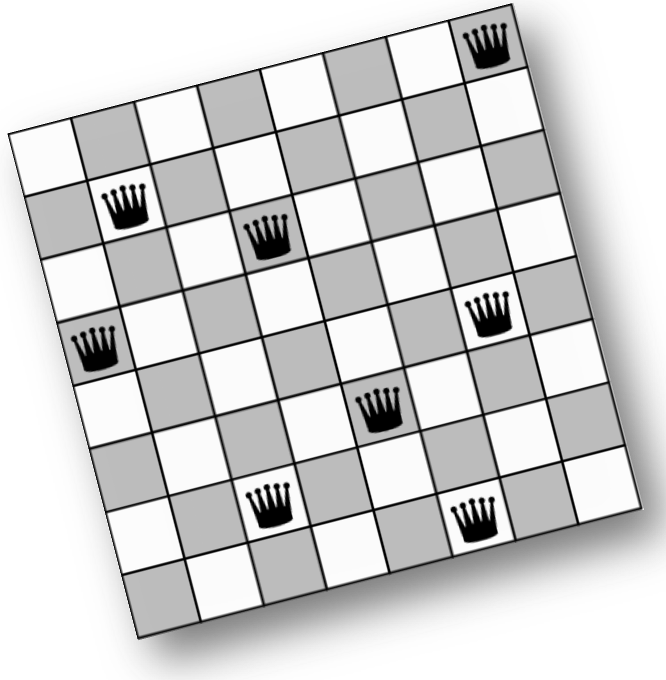
- **search** (backtracking)
- and **inference** (domain pruning) via **arc consistency** and **global constraints**



Let us assume a constraint model with Boolean variables **$B_{i,j}$** (domain {0,1}) describing whether some queen is at position (i,j).

The constraints may look like:

- exactly one queen at each column
  $\forall j: \sum_{i=1,..,n} B_{i,j} = 1$
- at most one queen at each row
  $\forall i: \sum_{j=1,..,n} B_{i,j} \leq 1$
- at most one queen at each diagonal
  $\forall k \in \{0,..,n-2\}: \sum_{l=1,..,n-k} B_{l,l+k} \leq 1$
  $\forall k \in \{1,..,n-2\}: \sum_{l=1,..,n-k} B_{l+k,l} \leq 1$
  $\forall k \in \{0,..,n-2\}: \sum_{l=1,..,n-k} B_{l,n-k-l+1} \leq 1$
  $\forall k \in \{1,..,n-2\}: \sum_{l=1,..,n-k} B_{l+k,n-l+1} \leq 1$



Now, the constraints can be decomposed as follows:

- $Sum(Xs) = 1 \Leftrightarrow at\_most\_one(Xs) \wedge at\_least\_one(Xs)$
- $Sum(Xs) \leq 1 \Leftrightarrow at\_most\_one(Xs)$
- $at\_least\_one(Xs) \Leftrightarrow \vee_i X_i$
- $at\_most\_one(Xs) \Leftrightarrow \wedge_{i<j} (\neg X_i \vee \neg X_j)$

The n-queens model can be expressed as a Boolean formula in a **conjunctive normal form** (and any satisfying assignment describes a solution).

Conjunctive normal form (CNF):

- **literal** is an atomic variable or its negation
- **clause** is a disjunction of literals
- **formula** in CNF is a conjunction of clauses

Example: $(\mathbf{A} \vee \neg \mathbf{B}) \wedge (\mathbf{B} \vee \neg \mathbf{C} \vee \neg \mathbf{D})$

Every sentence in propositional logic is logically equivalent to a conjunction of clauses.

Example:

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

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

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

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

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

Replace **a** $\Leftrightarrow$ **b** with $(\mathbf{a} \Rightarrow \mathbf{b} \wedge \mathbf{b} \Rightarrow \mathbf{a})$

Replace **a** $\Rightarrow$ **b** with $\neg\mathbf{a} \vee \mathbf{b}$

Apply De Morgan rules:

$\neg(\neg a) \equiv a$

$\neg(a \wedge b) \equiv (\neg a \vee \neg b)$

$\neg(a \vee b) \equiv (\neg a \wedge \neg b)$

How to efficiently find a satisfying assignment?

Combining search and inference, like in CSP.

Algorithm DPLL (Davis, Putnam, Logemann, Loveland)

– a sound and complete algorithm for verifying satisfiability of formulas in a CNF

```
function DPLL-SATISFIABLE?(s) returns true or false
  inputs: s, a sentence in propositional logic

  clauses ← the set of clauses in the CNF representation of s
  symbols ← 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 ← FIND-PURE-SYMBOL(symbols, clauses, model)
  if P is non-null then return DPLL(clauses, symbols – P, model ∪ {P=value})
  P, value ← FIND-UNIT-CLAUSE(clauses, model)
  if P is non-null then return DPLL(clauses, symbols – P, model ∪ {P=value})
  P ← FIRST(symbols); rest ← REST(symbols)
  return DPLL(clauses, rest, model ∪ {P=true}) or
         DPLL(clauses, rest, model ∪ {P=false}))
```

Early termination for partial models

- clause is true if any of its literals is true
- formula is not true if any of its clauses is not true

Pure symbol heuristics

- a pure symbol always appears with the same " sign" in all clauses
- the corresponding literal is set to true

Unit clause heuristics

- a unit clause is a clause with just one literal
- the corresponding literal is set to true

branching for backtracking

Component analysis

- If clauses can be separated into disjoint subsets not sharing a variable, the subsets can be solved independently.

Variable (and value) ordering

- **Degree heuristic** suggests choosing the variable that appears most frequently in the clauses.
- **Activity heuristic** suggests choosing the variable that appears most frequently in the conflicts.

Random restarts

- If there is no progress in search, restart with different random choices (for example, in variable selection) may help.

Clever indexing

- Efficient methods to identify, for example, unit clauses via so called **watched literals** (associate each clause with two literals and examine the clause only when any of these literals is assigned false).

Clause learning

- Analyze a conflict (failure during search) and encode the conflict as a new clause.

Can we exploit logical reasoning in construction of rational agents?

Logical methods can do reasoning about the world – we can deduce more information than that directly observable, via logical **inference**.

A knowledge-based agent uses a **knowledge base** – a set of sentences expressed in a given language – that can be updated by the operation **TELL** and can be queried about what is known using the operation **ASK**.

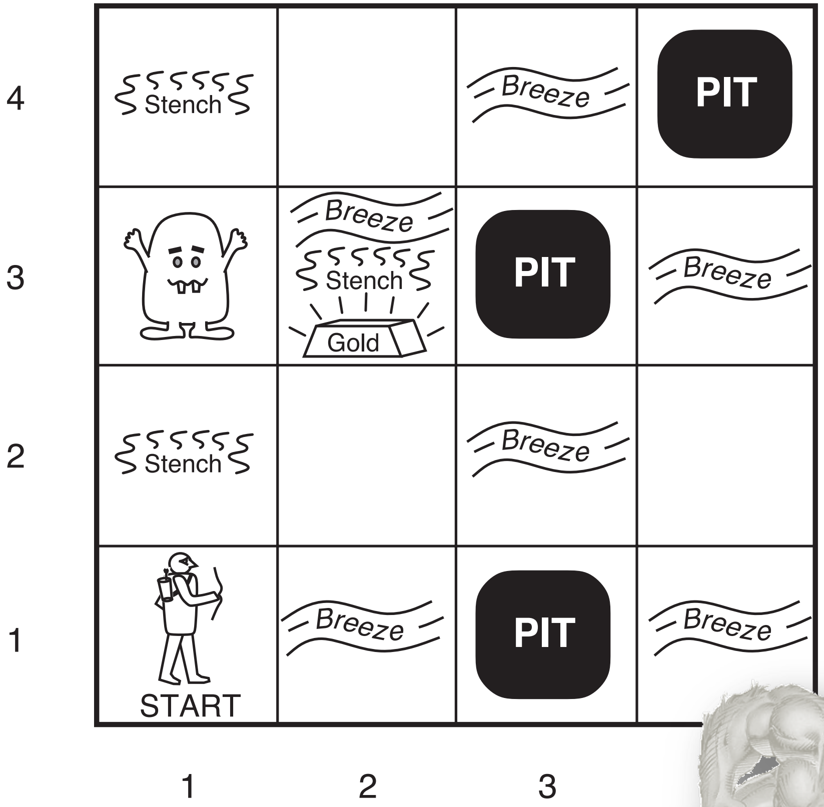
```
function KB-AGENT(percept) returns an action
   persistent: KB, a knowledge base
               t, a counter, initially 0, indicating time

   TELL(KB, MAKE-PERCEPT-SENTENCE(percept, t))
   action ← ASK(KB, MAKE-ACTION-QUERY(t))
   TELL(KB, MAKE-ACTION-SENTENCE(action, t))
   t ← t + 1
   return action
```

knowledge base contains information about observations as well as about own actions

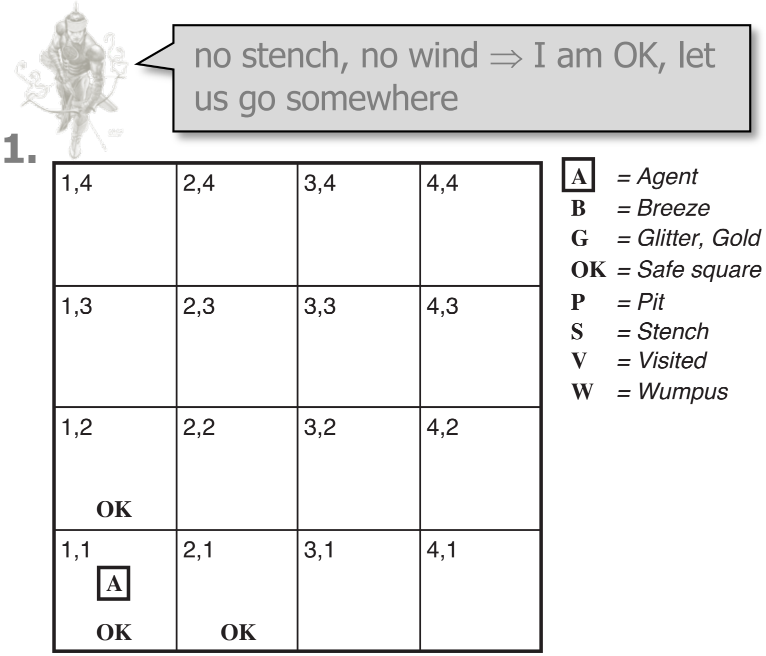
inference will help the agent to select an action even if information about the world is incomplete

A cave consisting of rooms connected by passageways, inhabited by the terrible **Wumpus**, a beast that eats anyone who enters its room, containing rooms with bottomless **pits** that will trap anyone, and a room with a heap of **gold**.

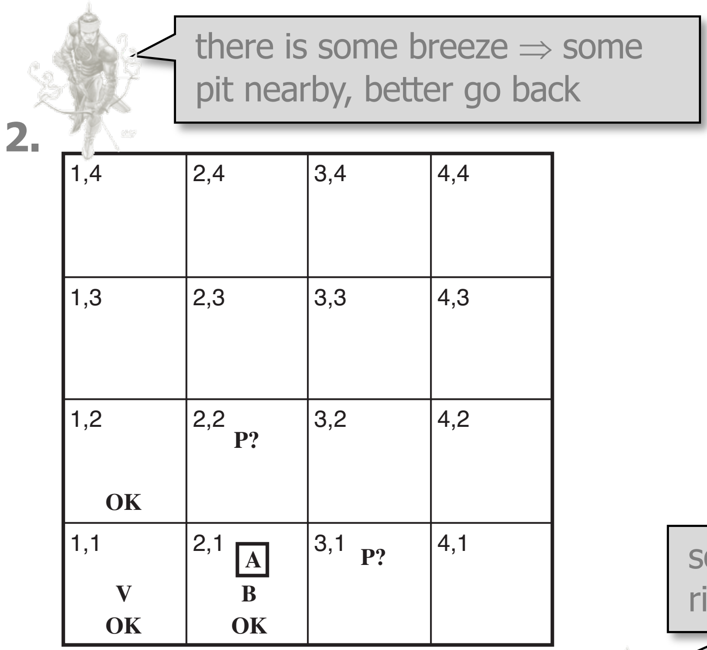


- The agent will perceive a **Stench** in the directly (not diagonally) adjacent squares to the square containing the Wumpus.
- In the squares directly adjacent to a pit, the agent will perceive a **Breeze**.
- In the square where the gold is, the agent will perceive a **Glitter**.
- When an agent walks into a wall, it will perceive a **Bump**.
- The Wumpus can be shot by an agent, but the agent has only one arrow.
  - Killed Wumpus emits a woeful **Scream** that can be perceived anywhere in the cave.

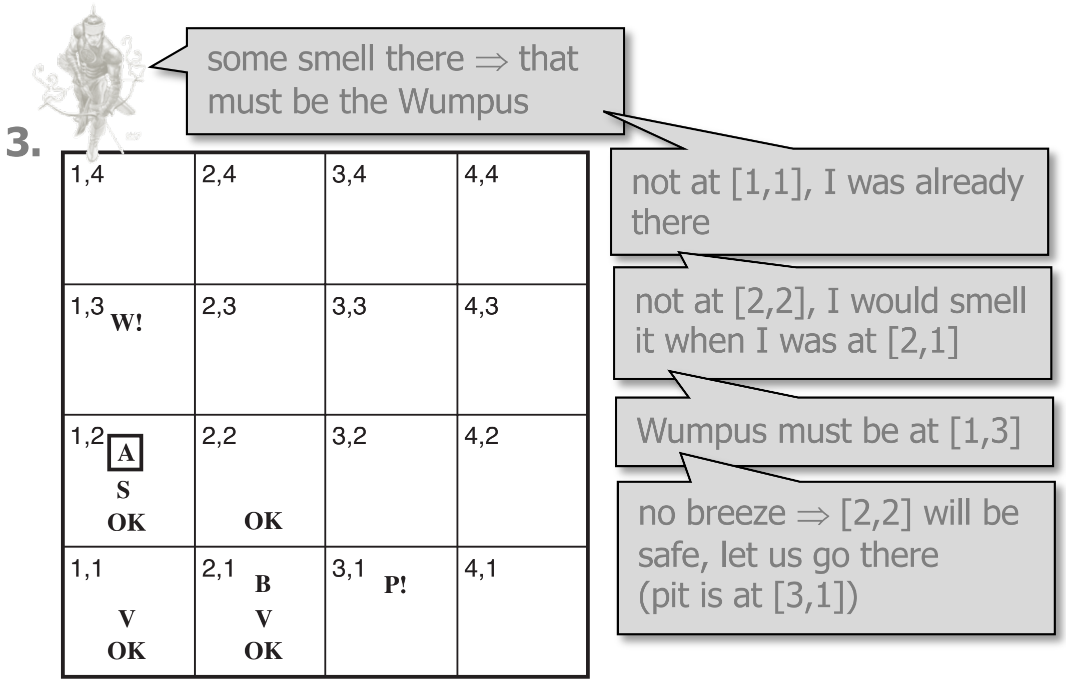
1.


no stench, no wind ⇒ I am OK, let us go somewhere
1,4
2,4
3,4
4,4
1,3
2,3
3,3
4,3
1,2
OK
2,2
3,2
4,2
1,1
A
OK
2,1
OK
3,1
4,1
A = Agent
B = Breeze
G = Glitter, Gold
OK = Safe square
P = Pit
S = Stench
V = Visited
W = Wumpus

2.

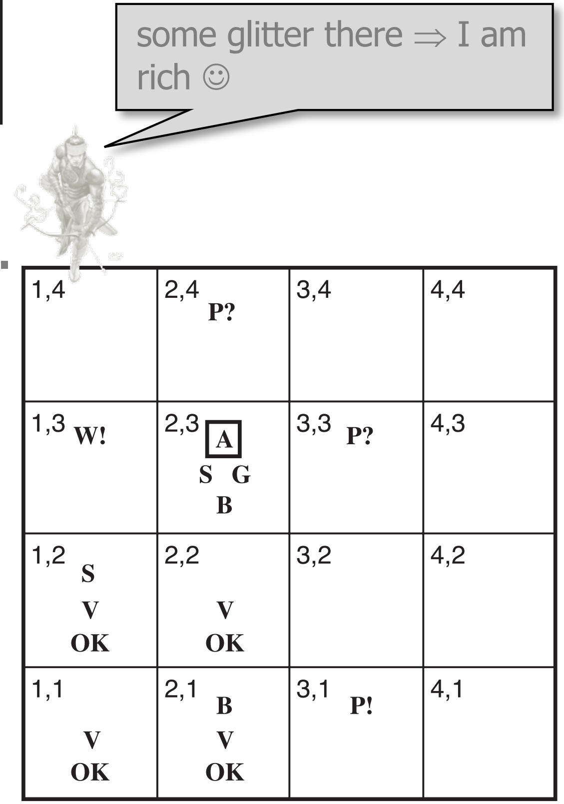

there is some breeze ⇒ some pit nearby, better go back
1,4
2,4
3,4
4,4
1,3
2,3
3,3
4,3
1,2
OK
2,2
P?
3,2
4,2
1,1
V
OK
2,1
A
B
OK
3,1
P?
4,1

3.

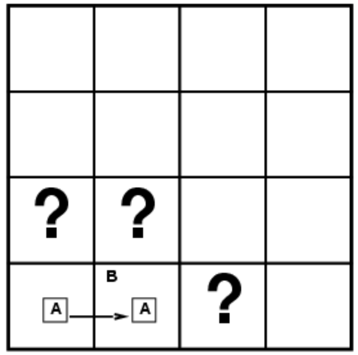

some smell there ⇒ that must be the Wumpus
not at [1,1], I was already there
not at [2,2], I would smell it when I was at [2,1]
Wumpus must be at [1,3]
no breeze ⇒ [2,2] will be safe, let us go there (pit is at [3,1])
1,4
2,4
3,4
4,4
1,3
W!
2,3
3,3
4,3
1,2
A
S
OK
2,2
OK
3,2
4,2
1,1
V
OK
2,1
B
V
OK
3,1
P!
4,1

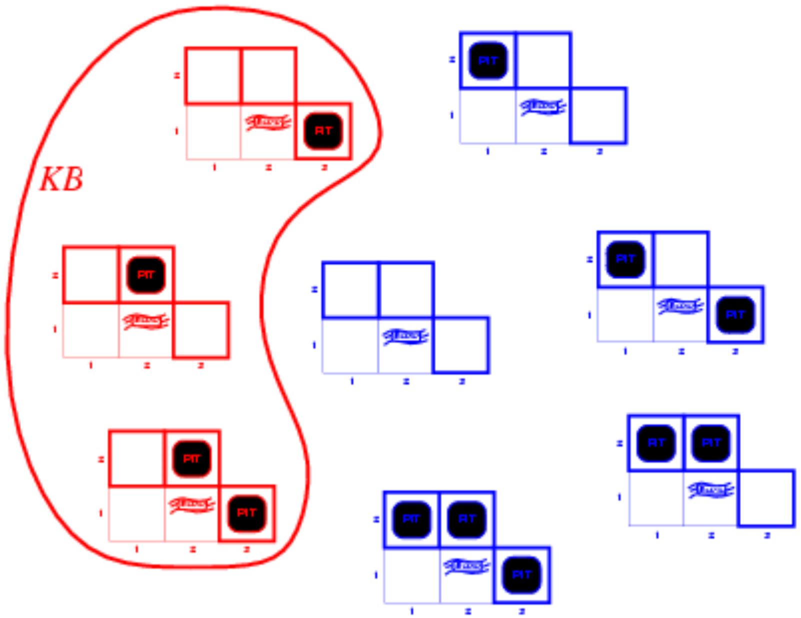
...

5.


some glitter there ⇒ I am rich ☺
1,4
2,4
P?
3,4
4,4
1,3
W!
2,3
A
S G
B
3,3
P?
4,3
1,2
S
V
OK
2,2
V
OK
3,2
4,2
1,1
V
OK
2,1
B
V
OK
3,1
P!
4,1

Assume a situation, when there is no percept at [1,1], we went right to [2,1] and feel Breeze there.





- For pit detection we have 8 (=2^3) possible **models** (states of the neighbouring world).
- Only three of these models correspond to our knowledge base, the other models conflict the observations:
  - no percept at [1,1]
  - Breeze at [2,1]

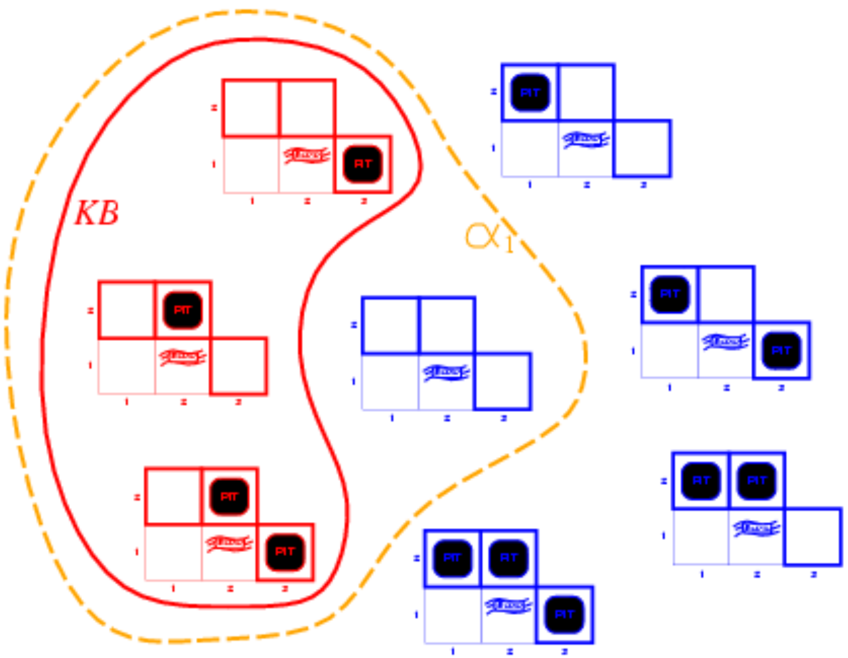
Let us ask whether the room [1,2] is safe.

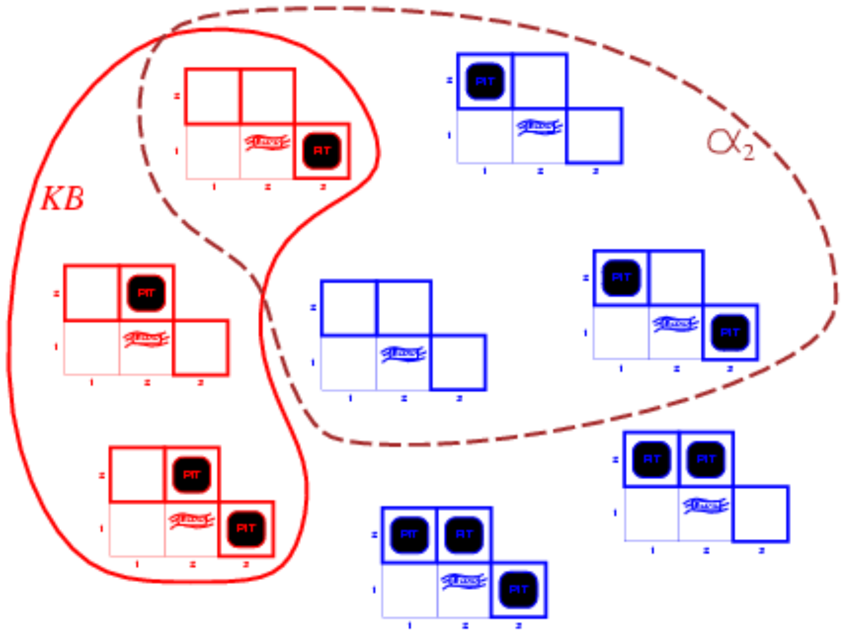
Is information α_1 = "[1,2] is safe" entailed by our representation?

- we compare models for KB and for α_1
- every model of KB is also a model for α_1 so α_1 is entailed by KB

And what about the room [2,2]?

- we compare models for KB and for α_2
- some models of KB are not models of α_2
- α_2 is not entailed by KB and we do not know for sure if room [2,2] is safe





Wumpus agent: logical perspective

Can we encode reasoning about the Wumpus world formally?

Possible models of the world correspond to **satisfying assignment** of a logical formula.

- known information about the world
  - $\neg \mathbf{P}_{1,1}$ no pit at [1, 1] (we are there)
  - $\neg \mathbf{W}_{1,1}$ no Wumpus at [1, 1] (we are there)
- observations
  - $\neg \mathbf{B}_{1,1}$ no Breeze at [1, 1]
  - $\mathbf{B}_{2,1}$ Breeze at [2, 1]
- we also know why and where breeze appears (model of world)
  - $\mathbf{B}_{x,y} \Leftrightarrow (\mathbf{P}_{x,y+1} \vee \mathbf{P}_{x,y-1} \vee \mathbf{P}_{x+1,y} \vee \mathbf{P}_{x-1,y})$
- and why a smell is generated
  - $\mathbf{S}_{x,y} \Leftrightarrow (\mathbf{W}_{x,y+1} \vee \mathbf{W}_{x,y-1} \vee \mathbf{W}_{x+1,y} \vee \mathbf{W}_{x-1,y})$
- and finally one “hidden” information – there is a single Wumpus in the world
  - $\mathbf{W}_{1,1} \vee \mathbf{W}_{1,2} \vee \ldots \vee \mathbf{W}_{4,4}$
  - $\neg \mathbf{W}_{1,1} \vee \neg \mathbf{W}_{1,2}$
  - $\neg \mathbf{W}_{1,1} \vee \neg \mathbf{W}_{1,3}$
  - **…**

$\mathbf{P}_{i,j}$ – pit at room (i,j)
$\mathbf{W}_{i,j}$ – Wumpus at room (i,j)
$\mathbf{B}_{i,j}$ – breeze at room (i,j)
$\mathbf{S}_{i,j}$ – stench at room (i,j)

Queries can ask whether a given cell is safe.

M (an assignment of truth values to all propositional variables) is a **model** of sentence α, if α is true in M.

– The set of models for α is denoted M(α).

P	Q	$\neg P$	$P \land Q$	$P \lor Q$	$P \Rightarrow Q$	$P \Leftrightarrow Q$
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*

Entailment: KB $\models \alpha$

means that α is a logical consequence of KB

– KB entails α iff M(KB) $\subseteq$ M(α)

Sentence (formula) is **satisfiable** if it is true in, or satisfied by, *some* model.

Example: A $\lor$ B, C

Sentence (formula) is **unsatisfiable** if it is not true in *any* model.

Example: A $\land \neg$A

Entailment can then be implemented as checking satisfiability as follows:

KB $\models \alpha$ if and only if **(KB $\land \neg\alpha$) is unsatisfiable.**

The resolution algorithm proves unsatisfiability of the formula (KB $\wedge \neg\alpha$) converted to a CNF. It uses a **resolution rule** that resolves two clauses with complementary literals (P and $\neg$P) to produce a new clause:

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

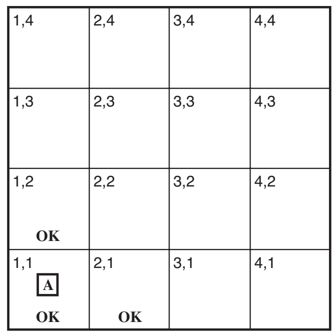
where l_i and m_j are the complementary literals

The algorithm stops when

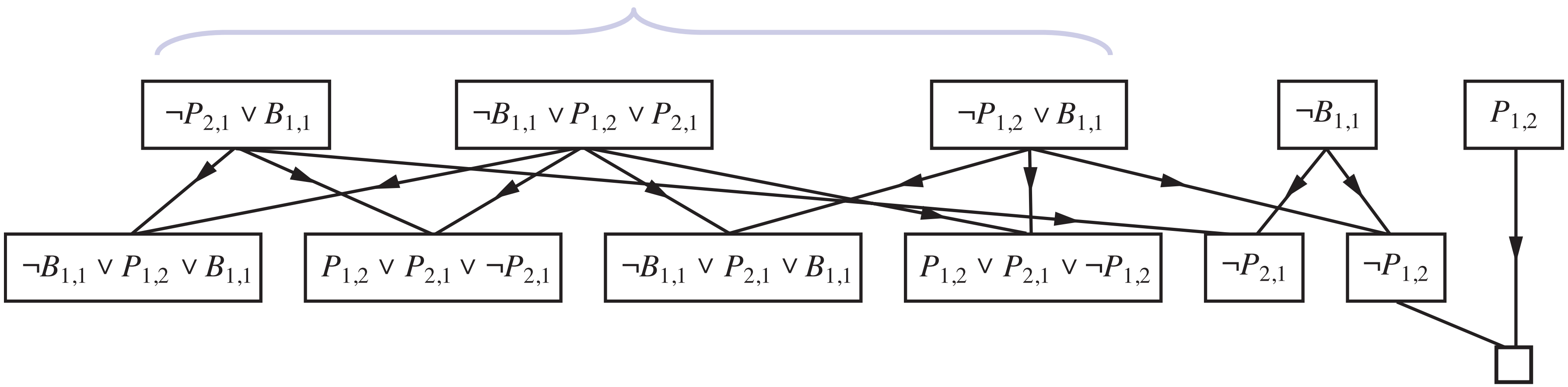
- no other clause can be derived (then $\neg$ KB $\models \alpha$)
- an empty clause was obtained (then KB $\models \alpha$)

Sound and complete algorithm

Example: Is cell (1,2) safe (no pit there)?



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



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

  clauses ← the set of clauses in the CNF representation of KB ∧ ¬α
  new ← { }
  loop do
     for each pair of clauses 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
```

For each pair of clauses with some complementary literals produce all possible resolvents.

an empty clause corresponds to false (an empty disjunction)

⇢ the formula ($KB \wedge \neg\alpha$) is unsatisfiable

⇢ α entailed by KB

All new resolvents are added to KB for next resolution.

we **reached a fixed point** (no new clauses added)

⇢ formula is satisfiable ($KB \wedge \neg\alpha$)

⇢ α is not entailed by KB

If the formula is satisfiable, how can we find its model?

take the symbols P_i one be one

1. if there is a clause with $\neg P_i$ such that the other literals are false with the current instantiation of $P_1,\dots,P_{i-1}$, then $\mathbf{P_i}$ **= false**
2. otherwise $\mathbf{P_i}$ **= true**

Many knowledge bases contain clauses of a special form – so called **Horn clauses**.

- Horn clause is a disjunction of literals of which at most one is positive

Example: $C \wedge (\neg B \vee A) \wedge (\neg C \vee \neg D \vee B)$

- Such clauses are typically used in knowledge bases with sentences in the form of an implication (for example Prolog without variables)

Example: $C \wedge (B \Rightarrow A) \wedge (C \wedge D \Rightarrow B)$

We will solve the problem if a given propositional symbol – **query** – can be derived from the knowledge base consisting of Horn clauses only.

- we can use a special variant of the resolution algorithm running in linear time with respect to the size of KB
- **forward chaining** (from facts to conclusions)
- **backward chaining** (from a query to facts)

From the known facts we derive all possible consequences using the Horn clauses until there are no new facts or we prove the query.

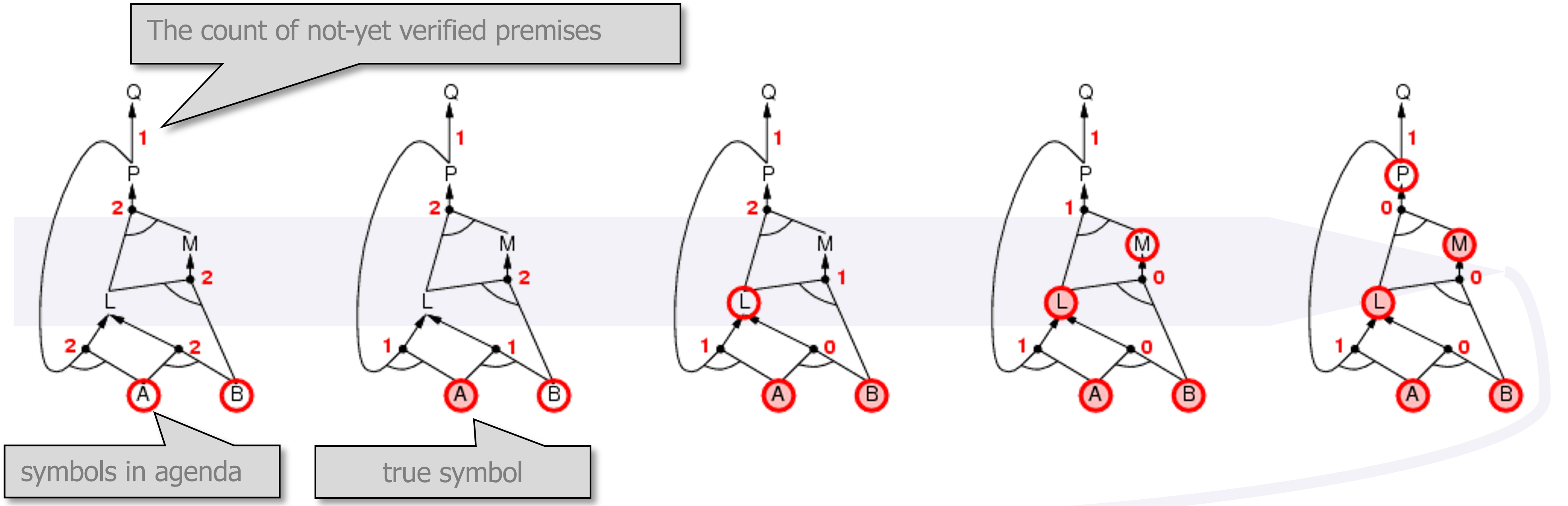
This is a **data-driven method**.

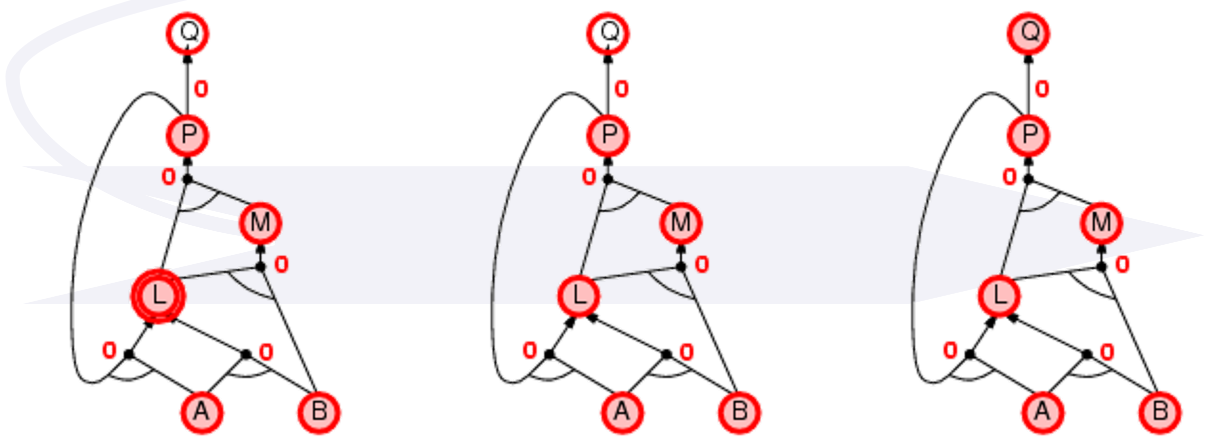
```
function PL-FC-ENTAILS?(KB, q) returns true or false
  inputs: KB, the knowledge base, a set of propositional definite clauses
          q, the query, a proposition symbol
  count ← a table, where count[c] is the number of symbols in c's premise
  inferred ← a table, where inferred[s] is initially false for all symbols
  agenda ← a queue of symbols, initially symbols known to be true in KB

  while agenda is not empty do
     p ← POP(agenda)
     if p = q then return true
     if inferred[p] = false then
        inferred[p] ← true
        for each clause c in KB where p is in c.PREMISE do
           decrement count[c]
           if count[c] = 0 then add c.CONCLUSION to agenda
  return false
```

For each clause we keep the number of not yet verified premises that is decreased when we infer a new fact. The clause with zero unverified premises gives a new fact (from the head of the clause).

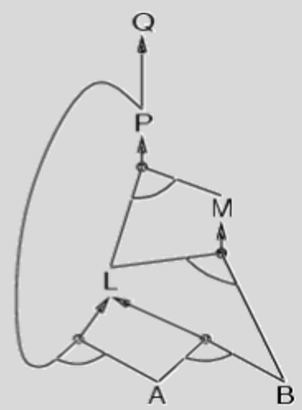
- **sound and complete** algorithm for Horn clauses
- **linear time complexity i**n the size of knowledge base





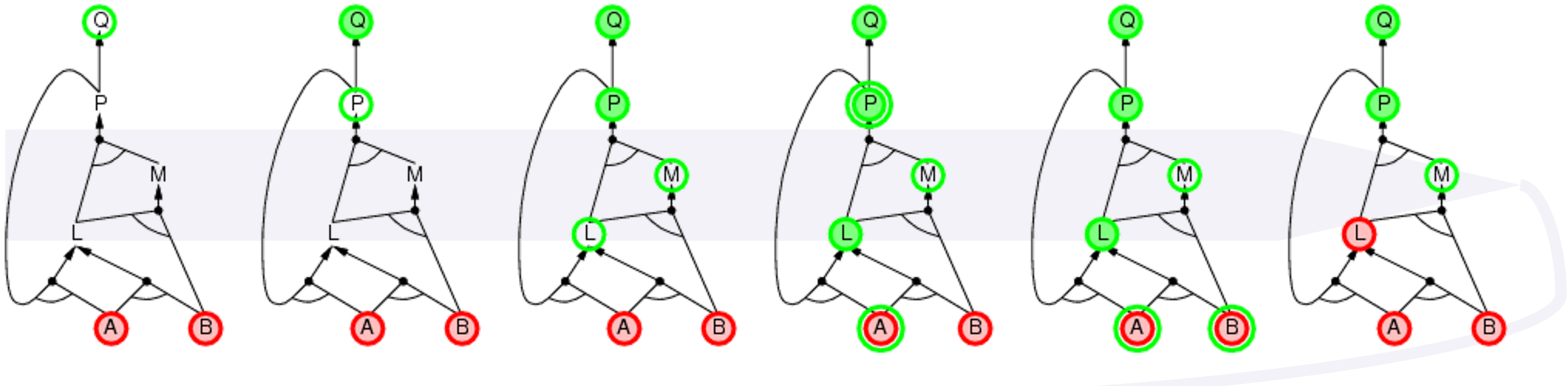
Knowledge base with a graphical representation

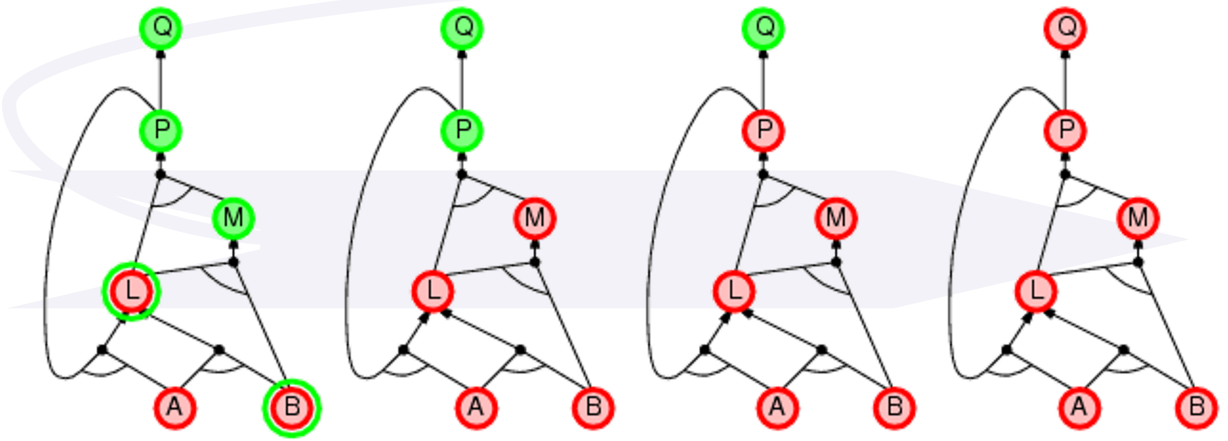
$P \Rightarrow Q$
$L \wedge M \Rightarrow P$
$B \wedge L \Rightarrow M$
$A \wedge P \Rightarrow L$
$A \wedge B \Rightarrow L$
A
B



The query is decomposed (via the Horn clause) to sub-queries until the facts from KB are obtained.

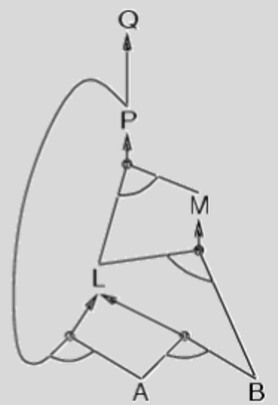
Goal-driven reasoning.





Knowledge base with a graphical representation

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



(Propositional) logic provides a formal framework for **knowledge representation** and **reasoning**.

Reasoning is realized via **logical inference** – deducing whether a logical formula is a logical consequence (entailed) of a knowledge base (a set of facts and axioms)

- enumeration methods
  - exploring (searching) possible models
  - **DPLL algorithm**
- theorem proving
  - symbolic methods
  - **resolution algorithm**
    - forward and backward chaining as special cases of resolution

Department of Theoretical Computer Science and Mathematical Logic
bartak@ktiml.mff.cuni.cz