Knowledge Representation & Reasoning COMP9016

Dr Ruairí O'Reilly
ruairi.oreilly@cit.ie

Logical Agents

CORK INSTITUTE OF TECHNOLOGY
INSTITIÚID TEICNEOLAÍOCHTA CHORCAÍ

Outline

"In which we design agents that can form representations of a complex world, use a process of inference to derive new representations about the world, and use these new representations to deduce what to do."

>> Knowledge-based agents

>> Wumpus world

>> Logic in general—models and entailment

>> Propositional (Boolean) logic

>> Equivalence, validity, satisfiability

>> Inference rules and theorem proving

    > forward chaining

    > backward chaining

    > resolution

CORK INSTITUTE OF TECHNOLOGY
INSTITIÚID TEICNEOLAÍOCHTA CHORCAÍ

Knowledge Based Agents

>> Central component of a knowledge-based agent is its **knowledge base**, or **KB**.

> **Knowledge base** = set of **sentences** in a formal language

> Each sentence is expressed in a language called a **knowledge representation language** and represents some assertion about the world (**axiom**).

>> Adding new sentences to the **KB**

> **TELL** and **ASK**

>> **Declarative approach** to building an agent (or other system):

Tell it what it needs to know

Then it can Ask itself what to do

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

Knowledge Based Agents

- Each time the agent program is called, it does three things:

1) First, it **TELLs** the knowledge base what it perceives.
2) Second, it **ASKs** the knowledge base what action it should perform.
3) Third, the agent program **TELLs** the knowledge base which action was chosen, and the agent executes the action

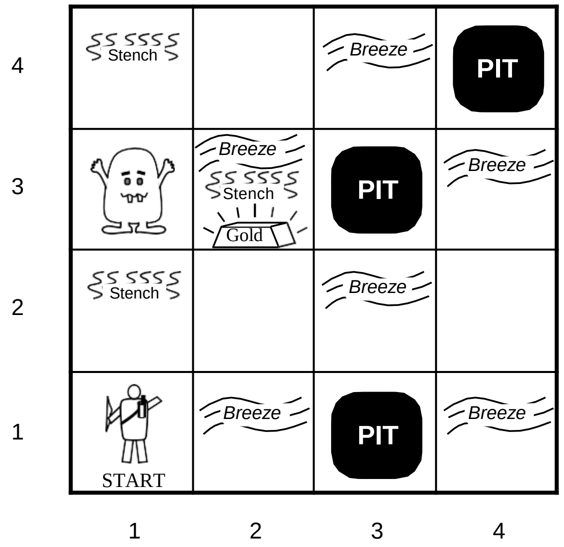
The agent must be able to?

```
function KB-Agent( percept) returns an action
   static: 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
```

Represent states, actions, etc. Incorporate new percepts, Update internal representations of the world, Deduce hidden properties of the world, Deduce appropriate actions

The Wumpus World

>> Text



Wumpus World with PEAS description

>> Performance measure

gold +1000, death -1000

-1 per step, -10 for using the arrow

>> Environment

Squares adjacent to wumpus are smelly

Squares adjacent to pit are breezy

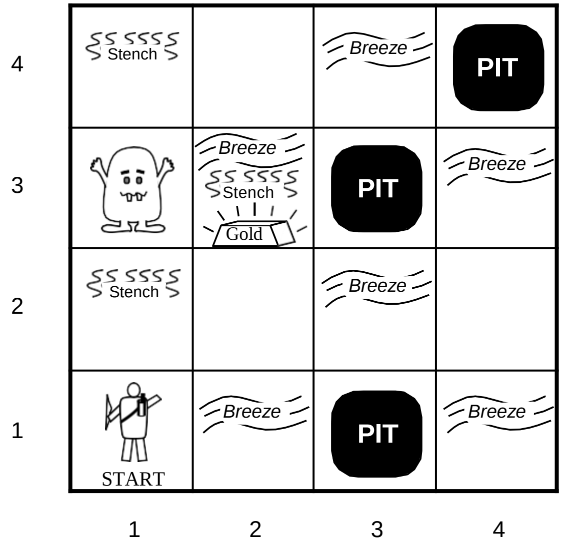
Glitter iff gold is in the same square

Shooting kills wumpus if you are facing it

Shooting uses up the only arrow

Grabbing picks up gold if in same square

Releasing drops the gold in same square



>> Actuators

Left turn, Right turn, Forward, Grab, Release, Shoot

>> Sensors

Breeze, Glitter, Smell

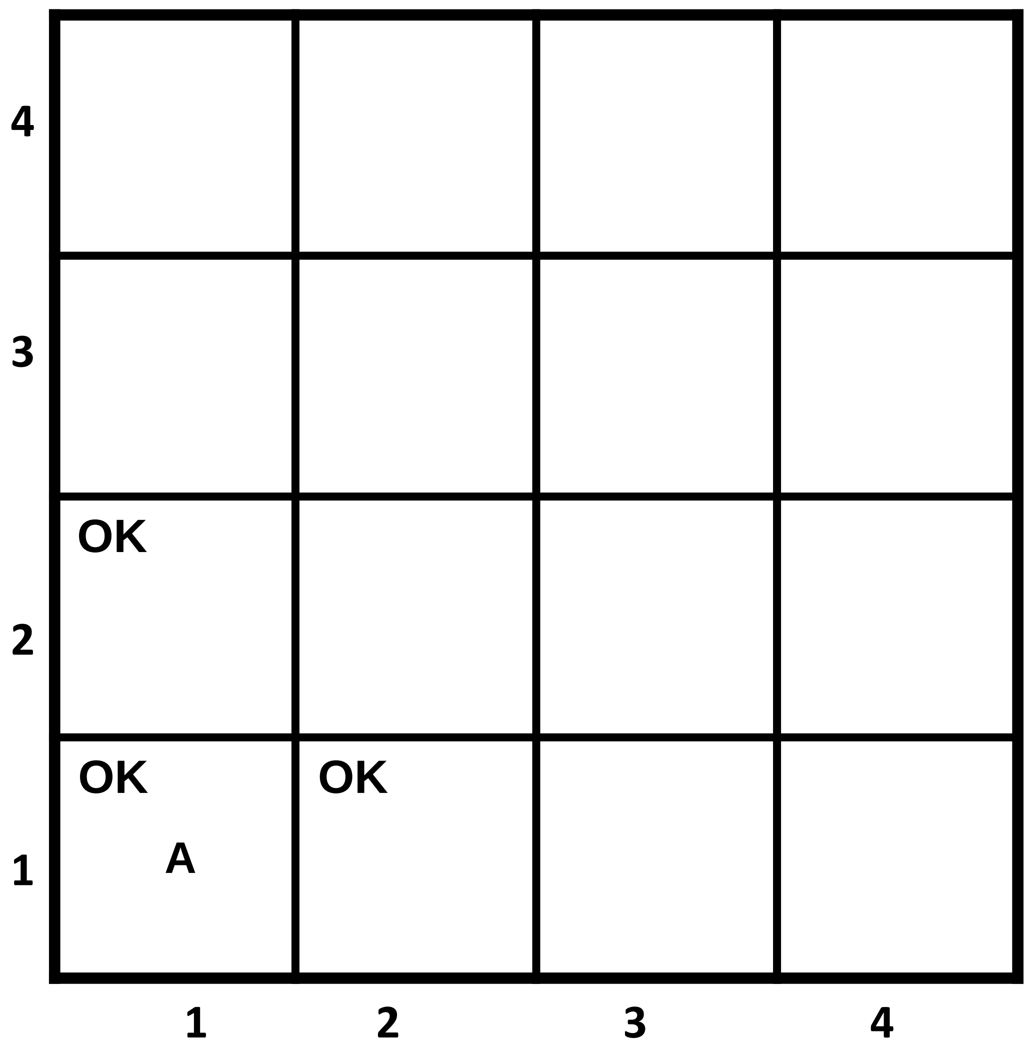
>> Percepts?

[Stench, Breeze, None, None, None]

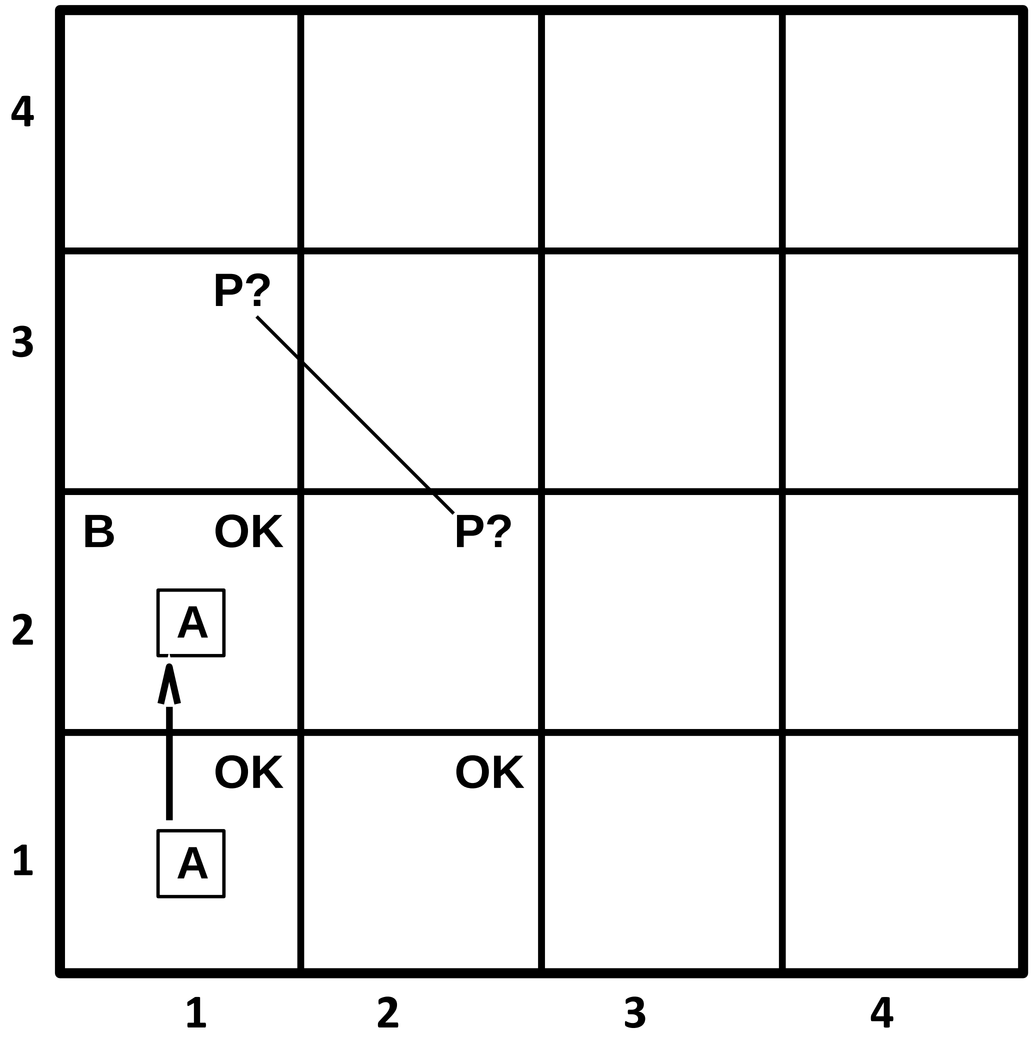
Wumpus world characterisation

>> Observable? No—only local perception
>> Deterministic? Yes—outcomes exactly specified
>> Episodic? No—sequential at the level of actions
>> Static? Yes—Wumpus and Pits do not move
>> Discrete? Yes
>> Single-agent? Yes—Wumpus is essentially a natural

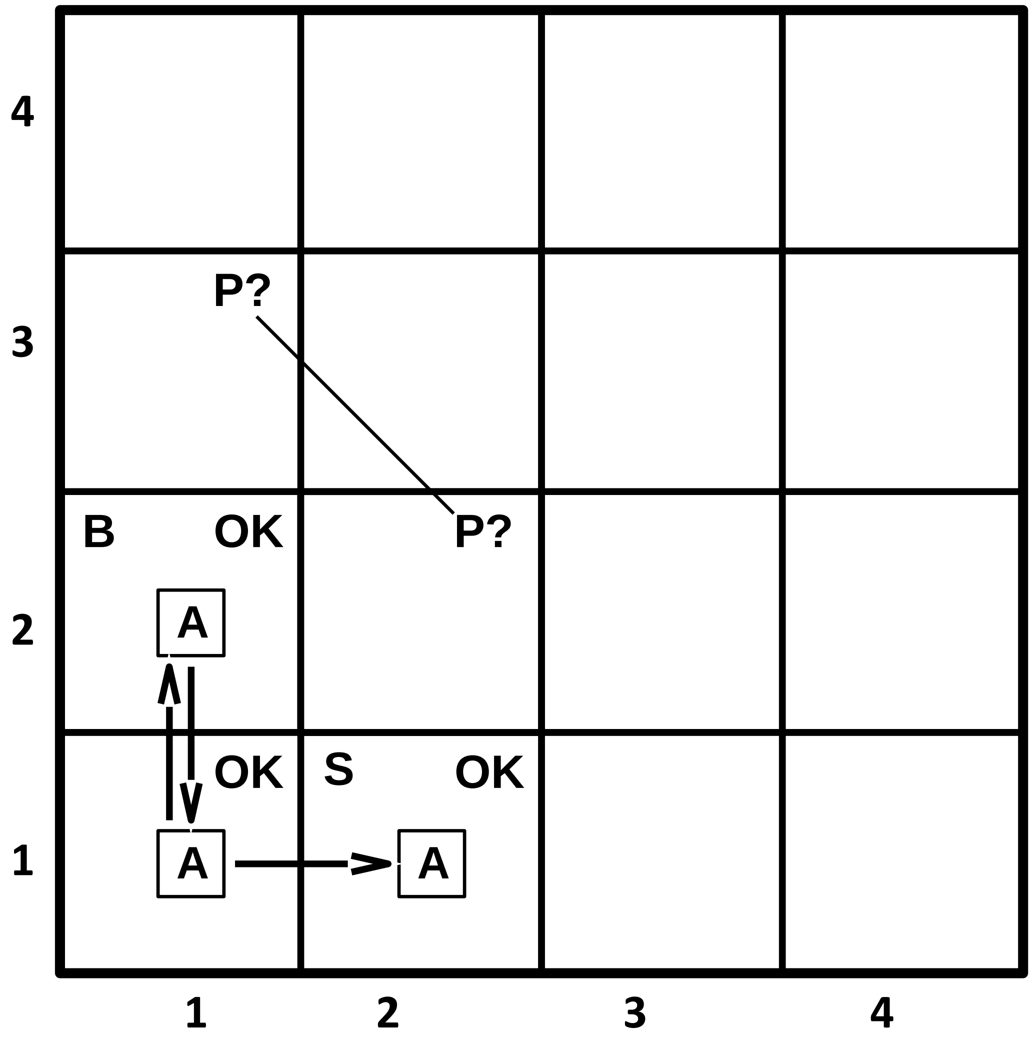
Exploring a wumpus world



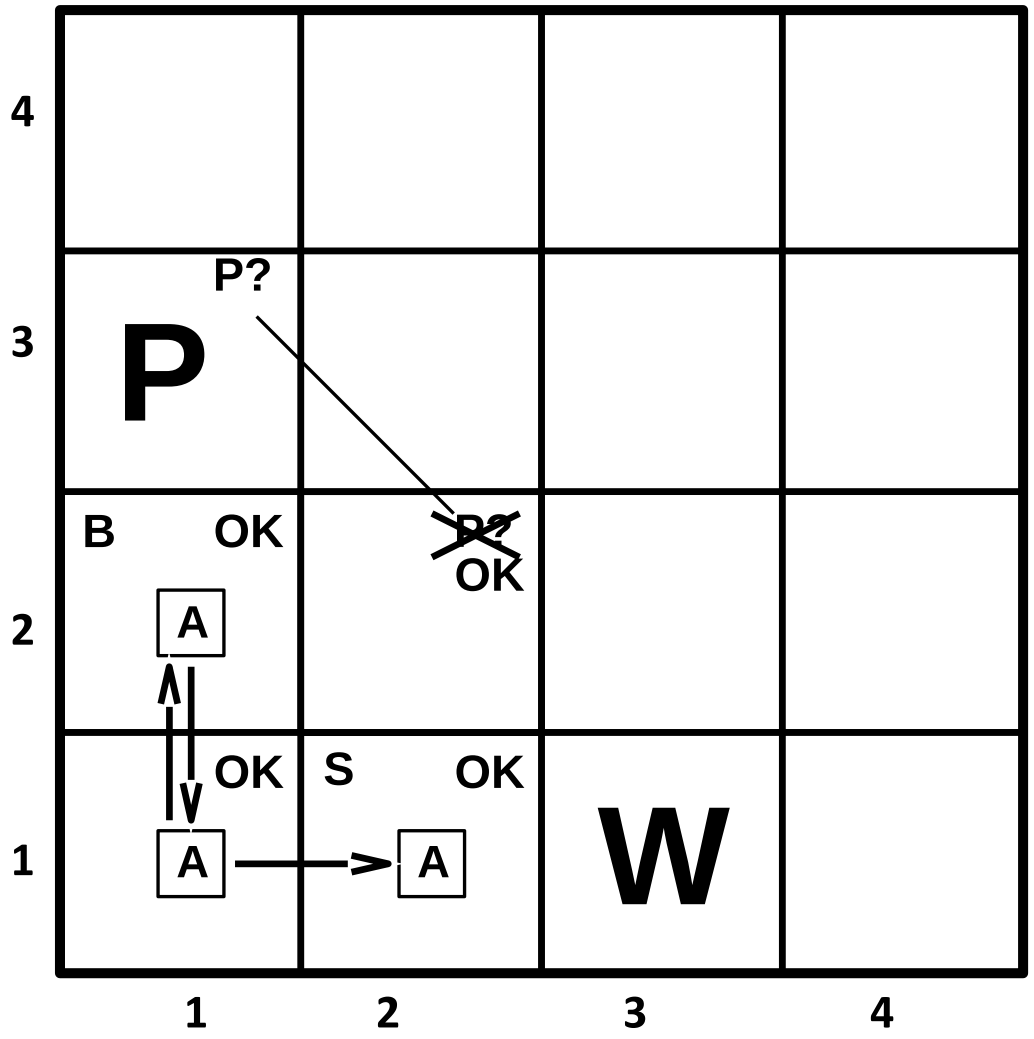
Exploring a wumpus world



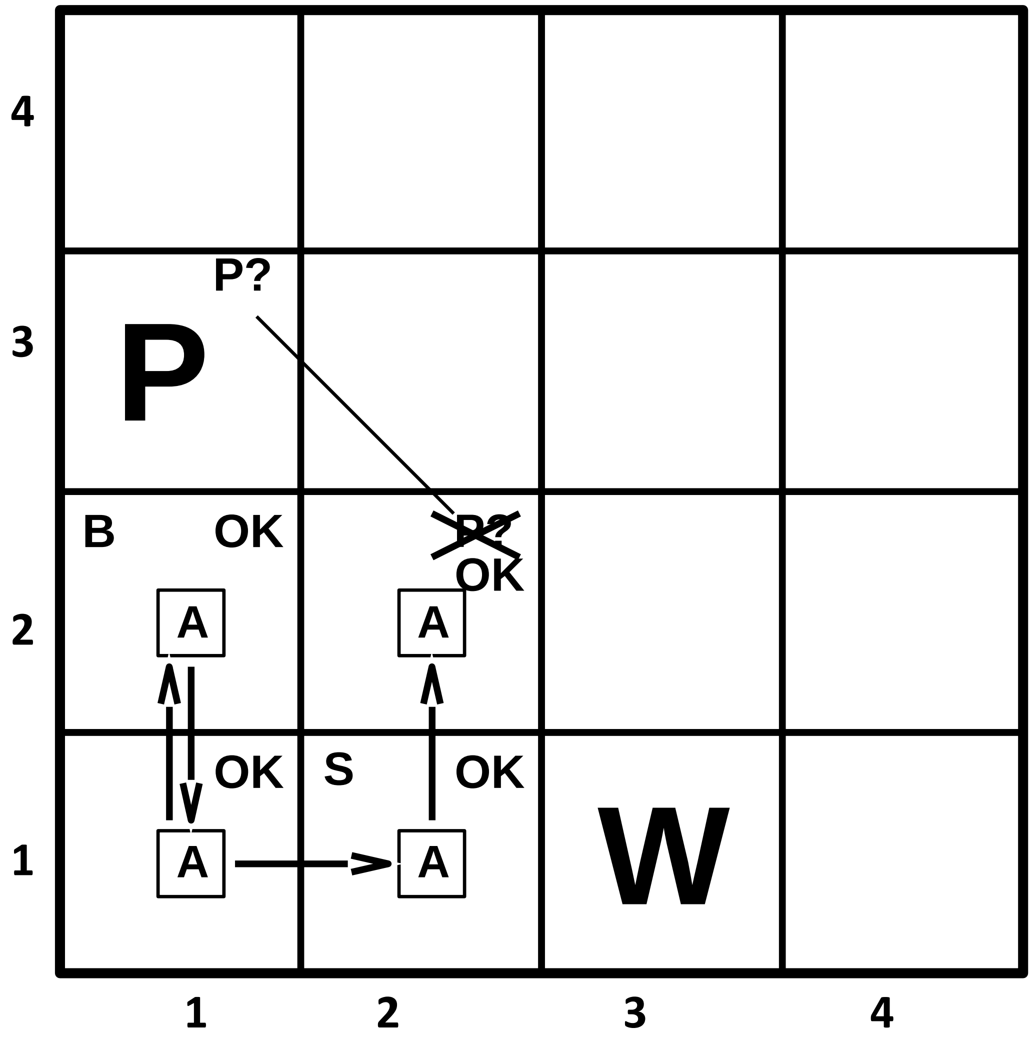
Exploring a wumpus world



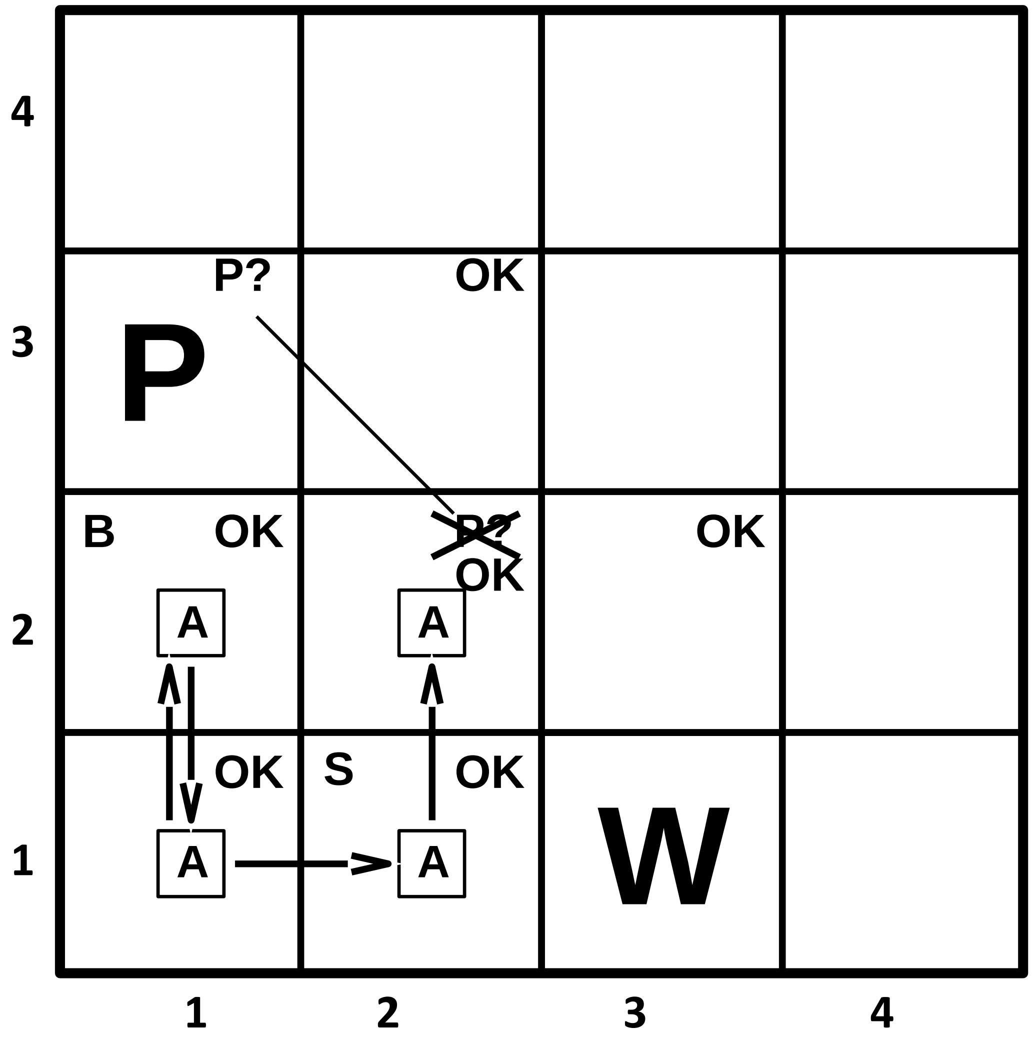
Exploring a wumpus world



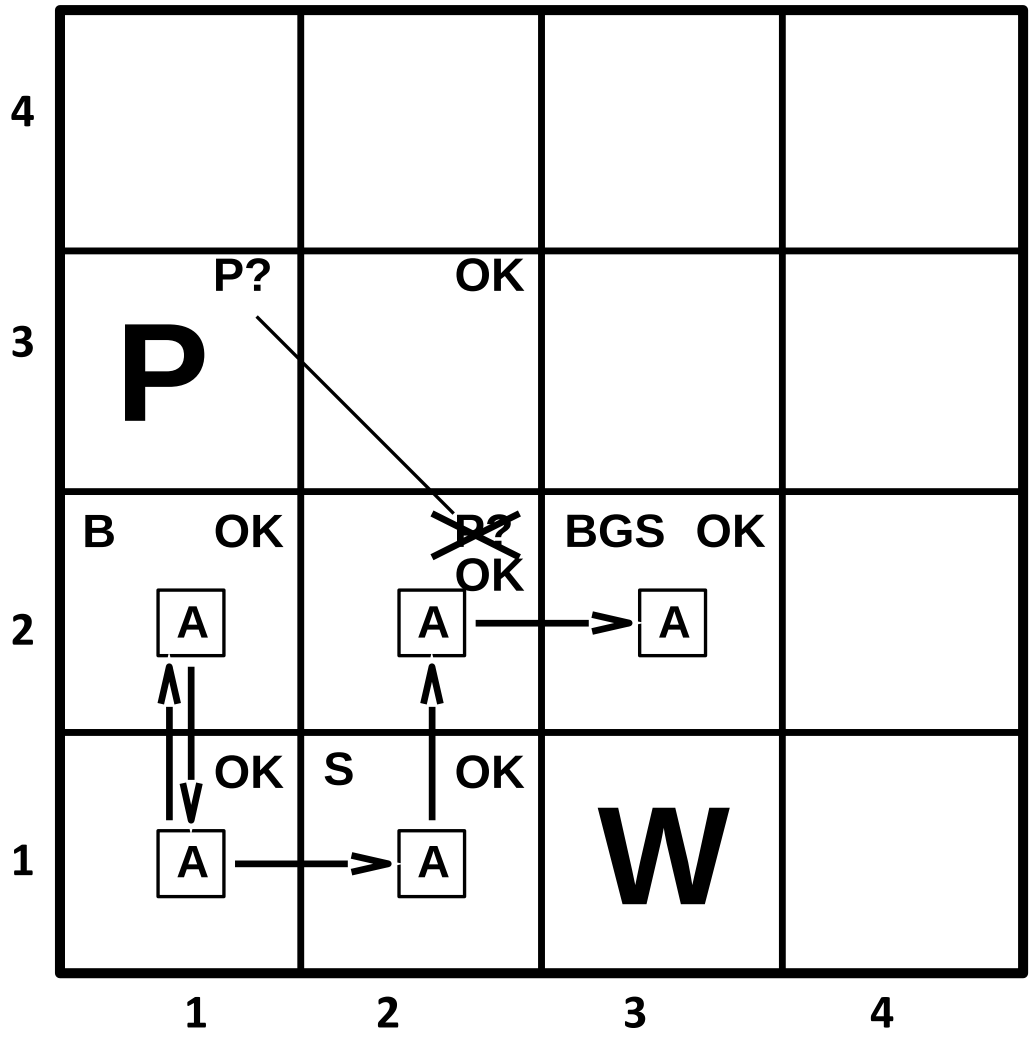
Exploring a wumpus world



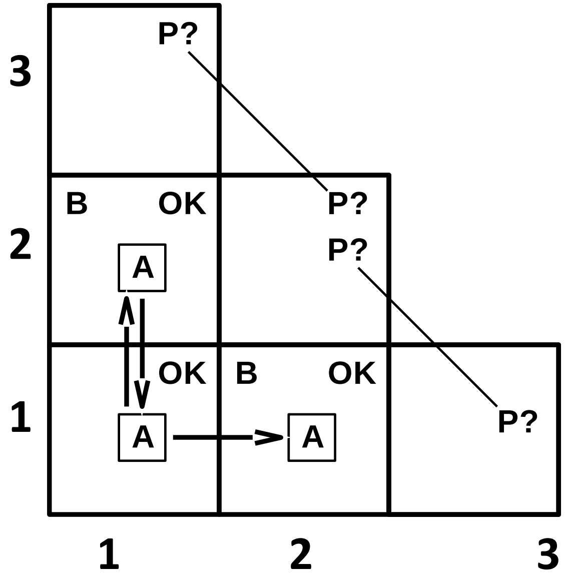
Exploring a wumpus world

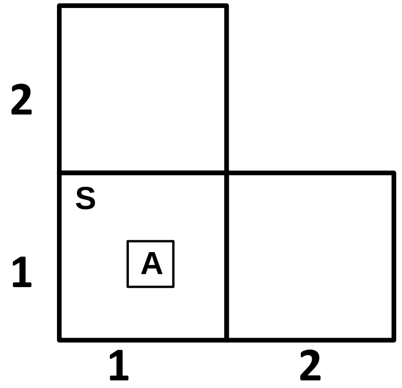


Exploring a wumpus world



Other tight spots





>> Breeze in (1,2) and (2,1)

⇒ no safe actions

Assuming pits uniformly distributed, (2,2) has pit w/ prob 0.86, vs. 0.31

Smell in (1,1)

⇒ cannot move

Can use a strategy of **coercion**: shoot straight ahead

wumpus was there ⇒ dead ⇒ safe

wumpus wasn't there ⇒ safe

CORK INSTITUTE OF TECHNOLOGY
INSTITIÚID TEICNEOLAÍOCHTA CHORCAÍ

Logic in general

>> **Logics** are formal languages for representing information such that conclusions can be drawn

>> **Syntax** defines the sentences in the language

>> **Semantics** define the "meaning" of sentences; i.e., define **truth** of a sentence in a world

E.g., the language of arithmetic

$x + 2 \geq y$ is a sentence; $x2 + y >$ is not a sentence

$x + 2 \geq y$ is true iff the number $x + 2$ is no less than the number y

$x + 2 \geq y$ is true in a world where $x = 7, y = 1$

$x + 2 \geq y$ is false in a world where $x = 0, y = 6$

Entailment

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

$KB \models \alpha$

Knowledge base **KB** entails sentence **α**

if and only if

α 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

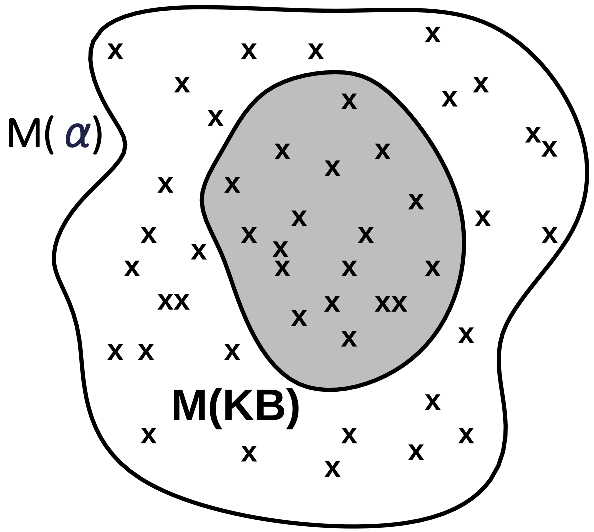
We say ***m*** is a model of a sentence **α** if **α** is true in

M (α) is the set of all models of **α**

Then ***KB |= α*** if and only if ***M (KB) ⊆ M (α)***

E.g. ***KB*** = Giants won and Reds won

α = Giants won



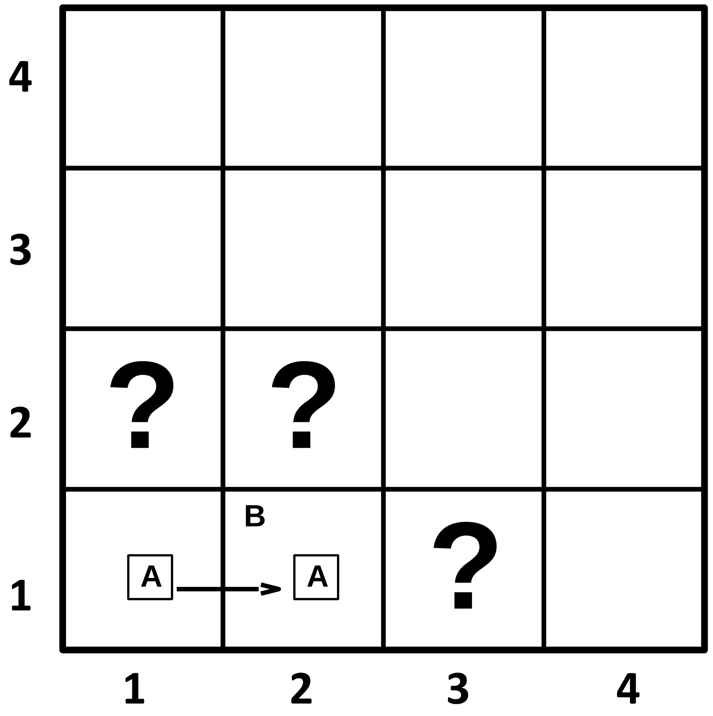
CORK INSTITUTE OF TECHNOLOGY
INSTITIÚID TEICNEOLAÍOCHTA CHORCAÍ

Entailment in the wumpus world

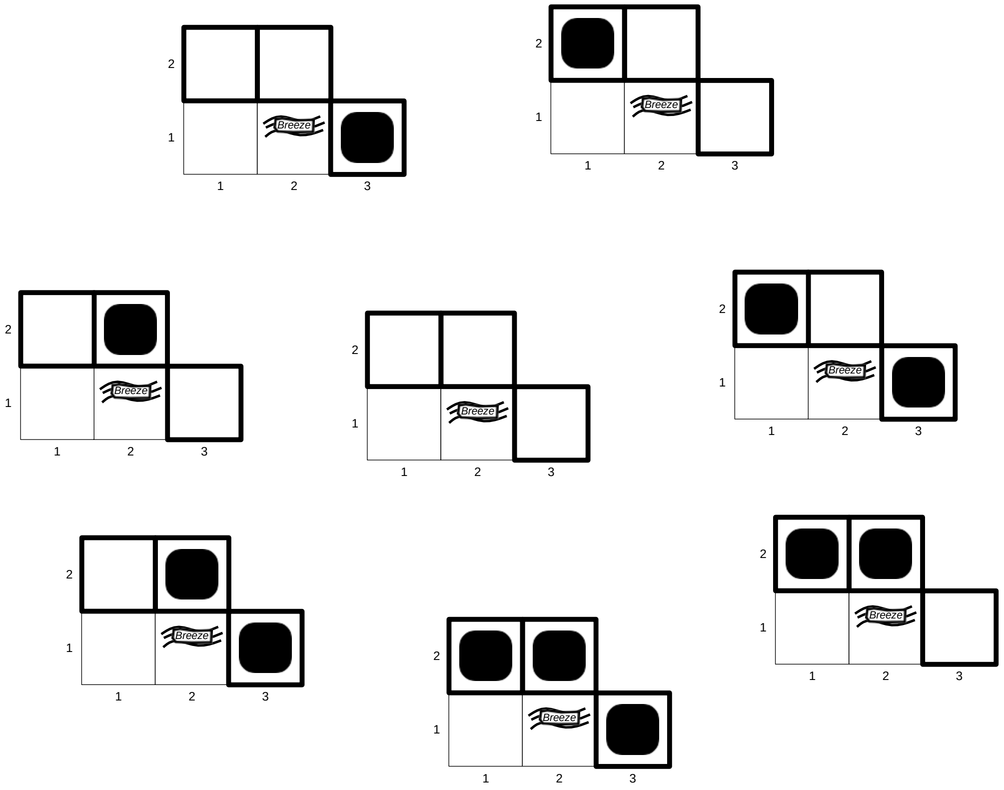
>> Situation after detecting nothing in [1,1], moving right, breeze in [2,1]

Consider possible models for ?s assuming only pits

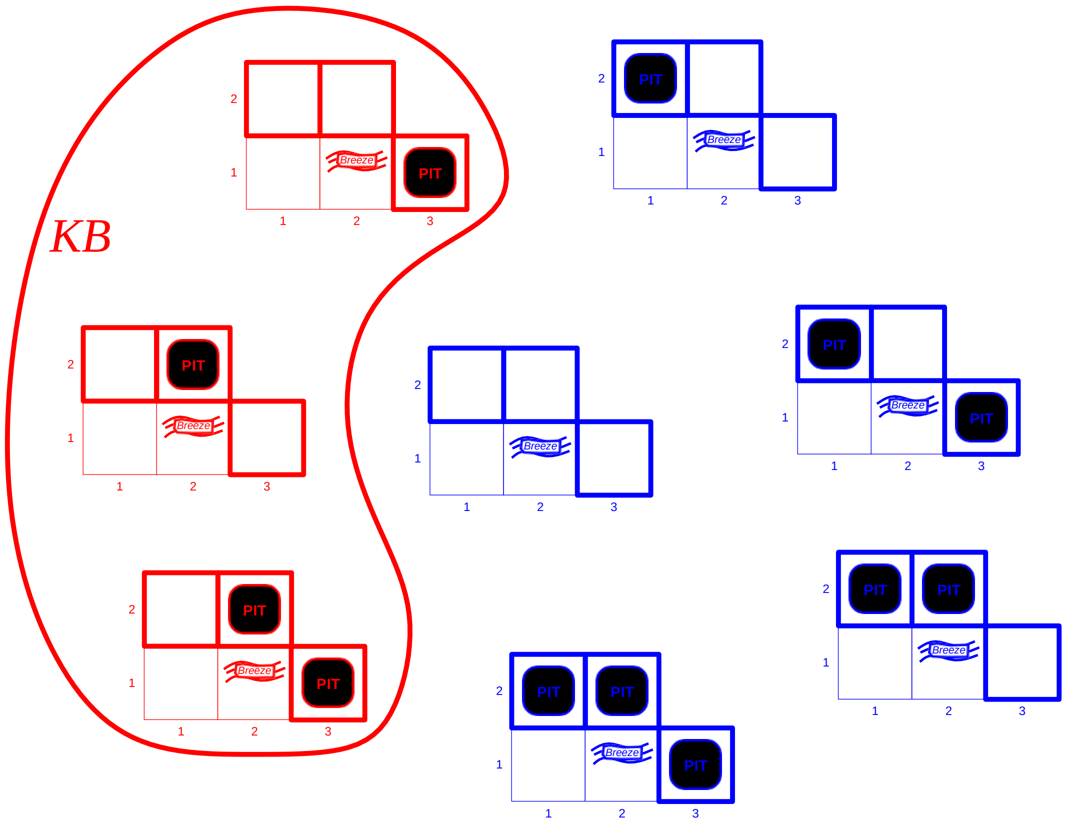
3 Boolean choices ⇒ 8 possible models



Wumpus models

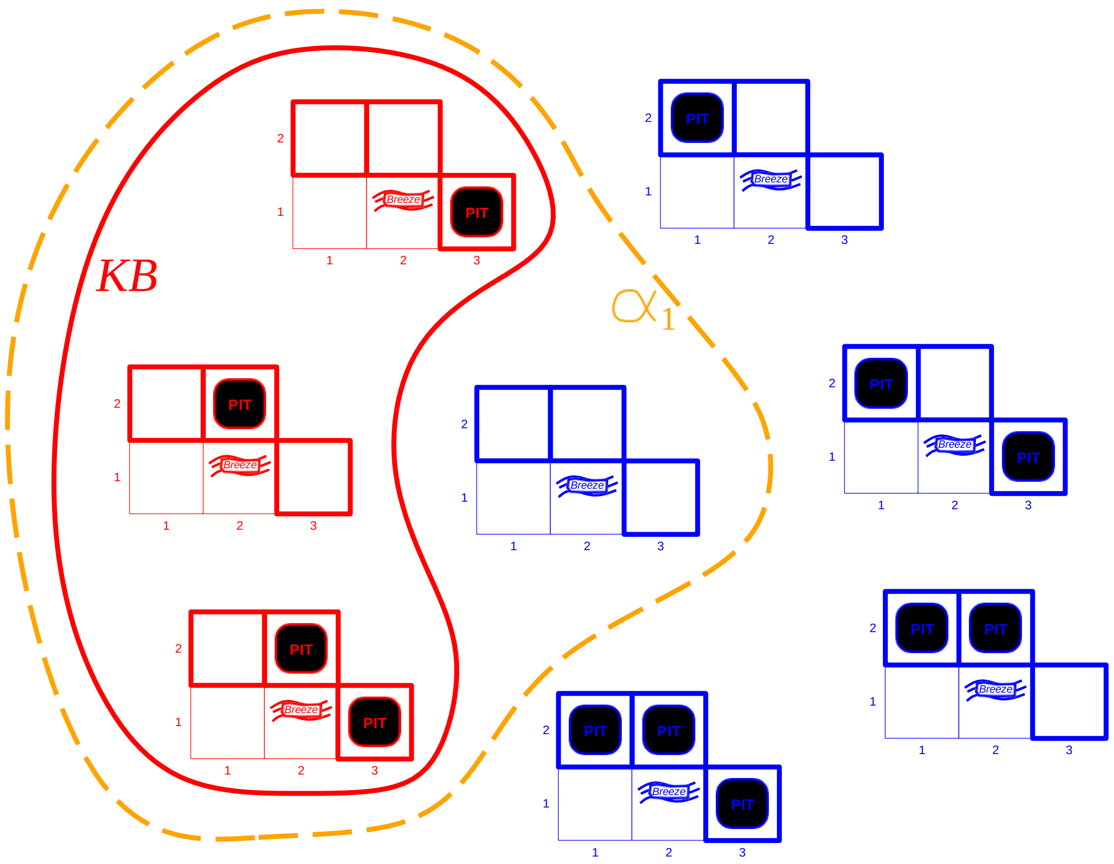


Wumpus models



>> ***KB*** = wumpus-world rules + observations

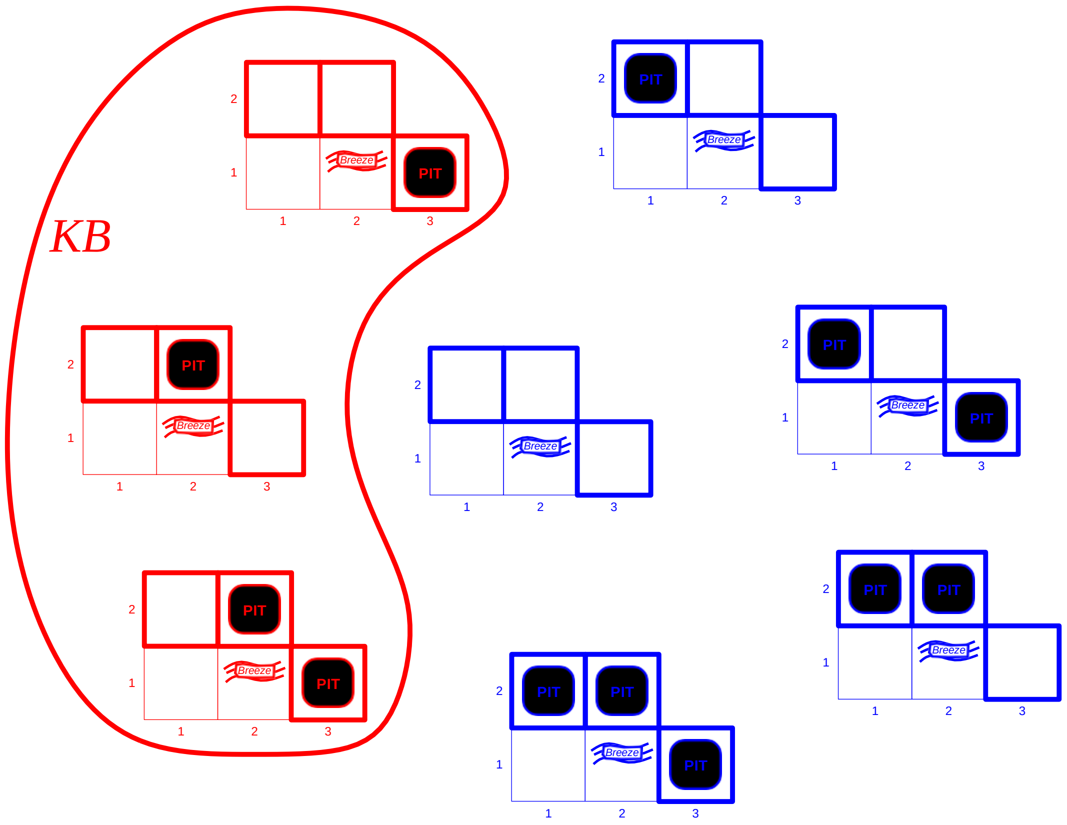
Wumpus models



>> ***KB*** = wumpus-world rules + observations

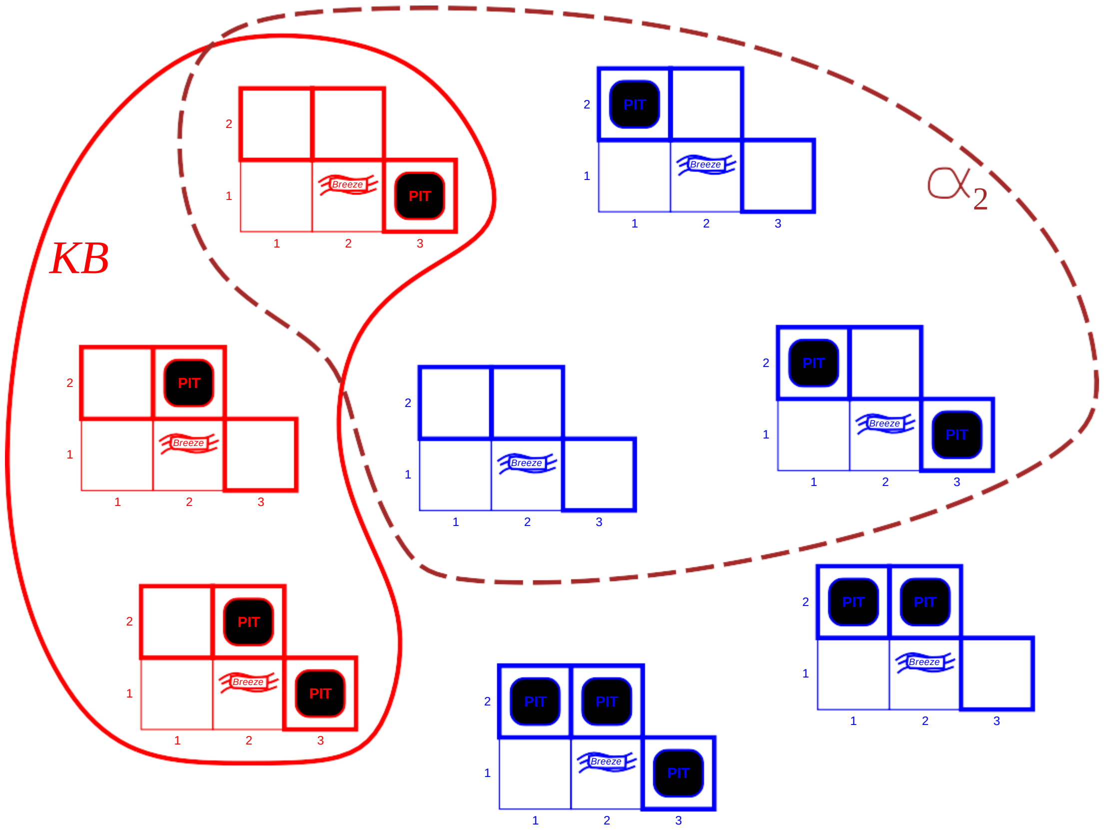
α1 = "[1,2] is safe", ***KB |= α1***, proved by ***model checking***

Wumpus models



>> ***KB*** = wumpus-world rules + observations

Wumpus models



>> ***KB*** = wumpus-world rules + observations

α2 = “[2,2] is safe”, ***KB |= α2***

Inference

>> $\boldsymbol{KB} \vdash_i \boldsymbol{\alpha}$ = sentence **α** can be derived from **KB** by procedure **i**

Consequences of **KB** are a haystack; **α** is a needle.

Entailment = needle in haystack; inference = finding it

> > **Soundness**: ***i*** is sound if

whenever $\boldsymbol{KB} \vdash_i \boldsymbol{\alpha}$, it is also true that $\boldsymbol{KB} \models \boldsymbol{\alpha}$

>> **Completeness**: ***i*** is complete if

whenever **KB |= α**, it is also true that KB $\vdash_i$ **α**

Preview: we will define a logic (FOL) 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: A very simple logic

>> Propositional logic is the simplest logic- illustrates basic ideas

The proposition symbols **P1, P2** etc are sentences

If **S** is a sentence, **¬S** is a sentence (negation)

If **S1** and **S2** are sentences, **S1 ∧ S2** is a sentence (conjunction)

If **S1** and **S2** are sentences, **S1 ∨ S2** is a sentence (disjunction)

If **S1** and **S2** are sentences, **S1 ⇒ S2** is a sentence (implication)

If **S1** and **S2** are sentences, **S1 ⇔ S2** is a sentence (biconditional)

Propositional logic: Semantics

Each model species true/false for each proposition symbol

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

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

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

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

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

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*

CORK INSTITUTE OF TECHNOLOGY
INSTITIÚID TEICNEOLAÍOCHTA CHORCAÍ

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"

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

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

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

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	*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*

>> Enumerate rows (different assignments to symbols),

if **KB is true in row, check that α is too**

Inference by enumeration

>> Depth-first enumeration of all models 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))
```

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

CORK INSTITUTE OF TECHNOLOGY
INSTITIÚID TEICNEOLAÍOCHTA CHORCAÍ

Logical equivalence

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

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

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

CORK INSTITUTE OF TECHNOLOGY
INSTITIÚID TEICNEOLAÍOCHTA CHORCAÍ

Validity and satisfiability

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

e.g., True, $A \lor \neg A$, $A \Rightarrow A$, $(A \land (A \Rightarrow B)) \Rightarrow B$

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

KB |= α if and only if (KB ⇒ α) is valid

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

A sentence is **unsatisfiable** if it is true in no models e.g., **A ∧ ¬A**

Satisfiability is connected to inference via the following:

KB |= α if and only if **(KB ∧ ¬α)** is unsatisfiable

i.e., prove **α** by ***reductio ad absurdum***

Proof methods

>> *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., $C \wedge (B \Rightarrow A) \wedge (C \wedge D \Rightarrow B)$

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

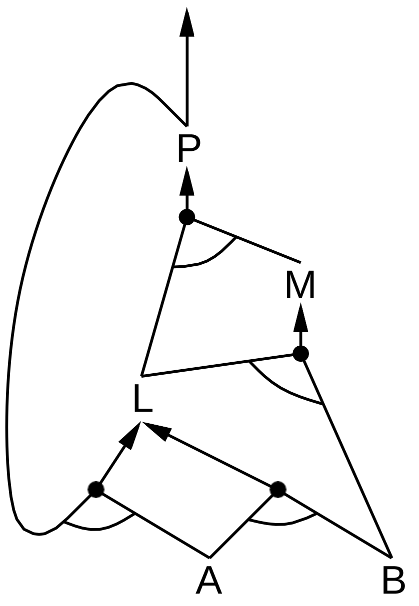
$$\frac{\alpha_1, \ldots, \alpha_n, \qquad \alpha_1 \wedge \cdots \wedge \alpha_n \Rightarrow \beta}{\beta}$$

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

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

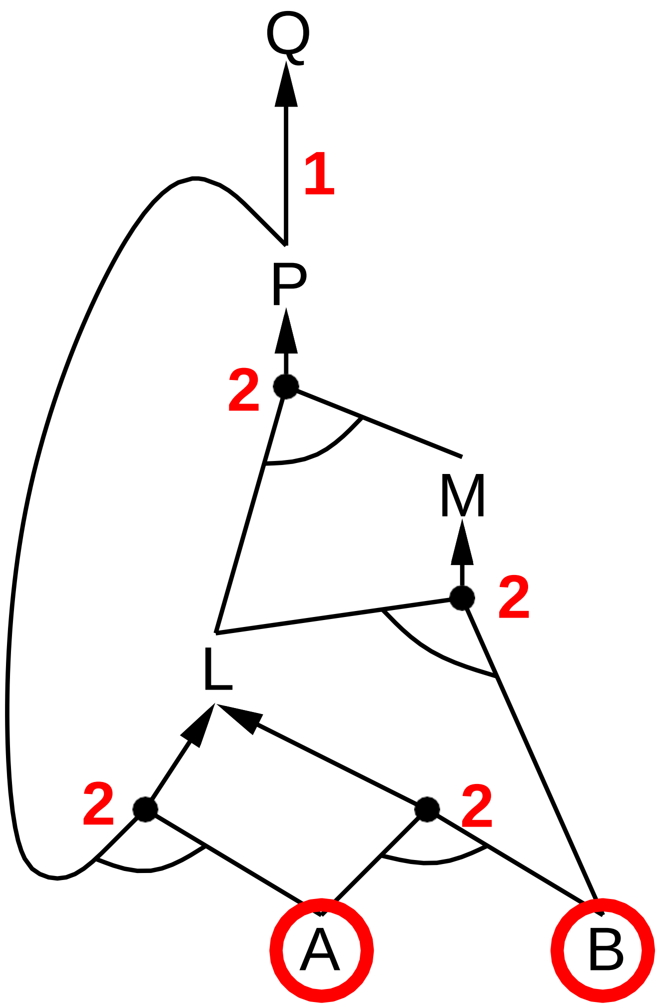


Forward chaining algorithm

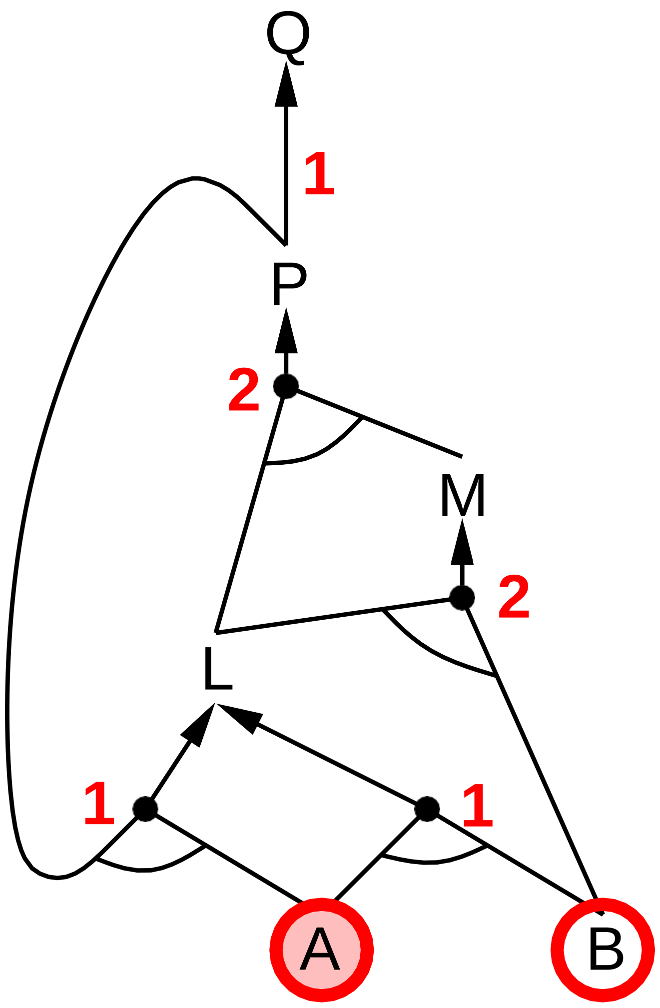
```
function PL-FC-Entails?(KB, q) returns true or false
    inputs: KB, the knowledge base, a set of propositional Horn clauses
            q, the query, a proposition symbol
   local variables: count, a table, indexed by clause, initially the number of premises
                    inferred, a table,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

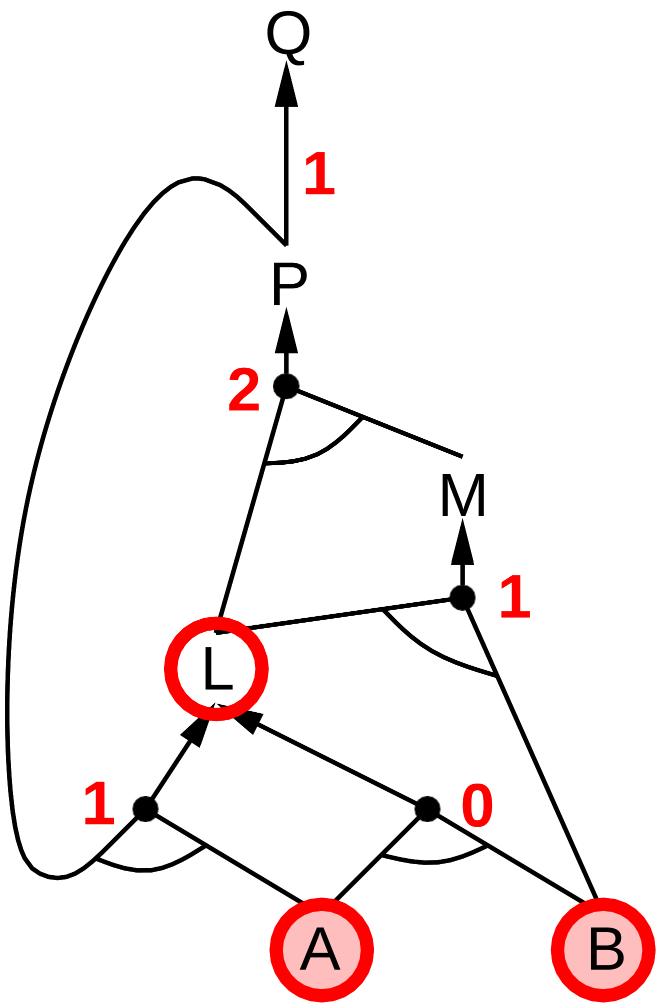
Forward chaining example



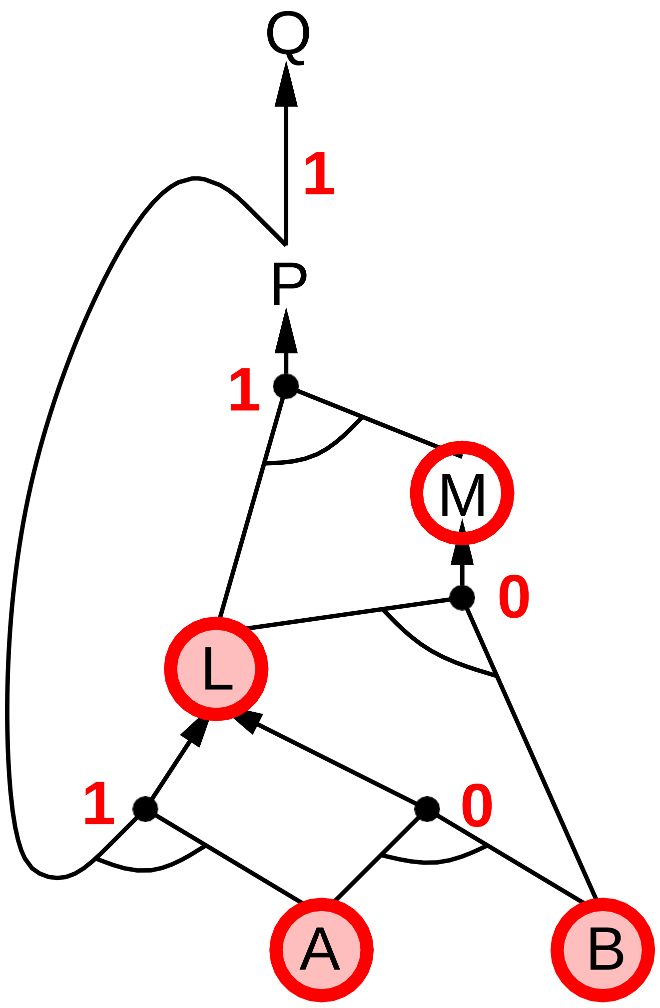
Forward chaining example



Forward chaining example

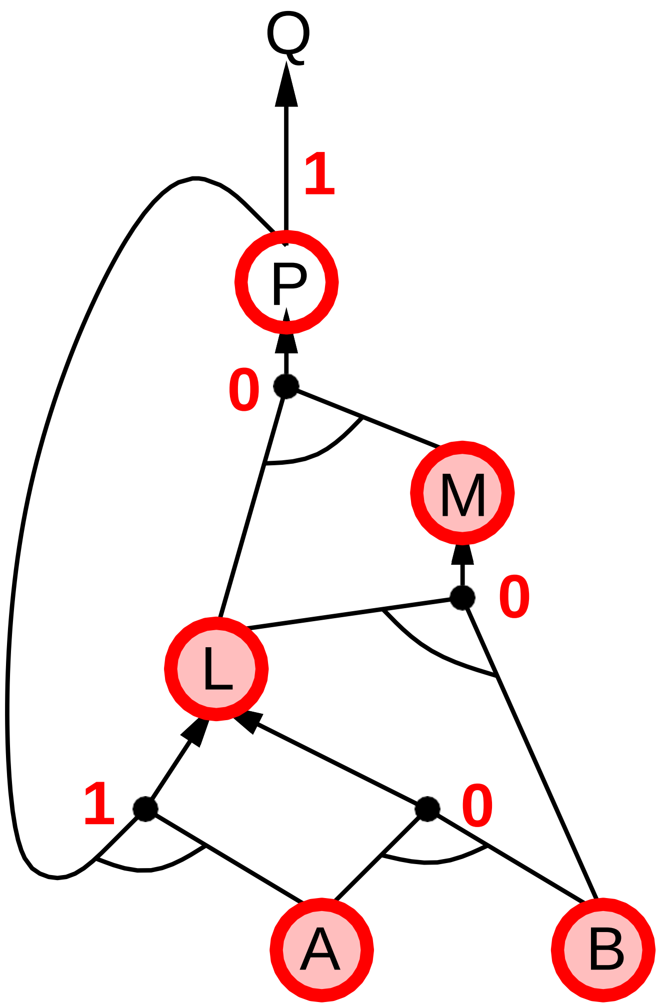


Forward chaining example

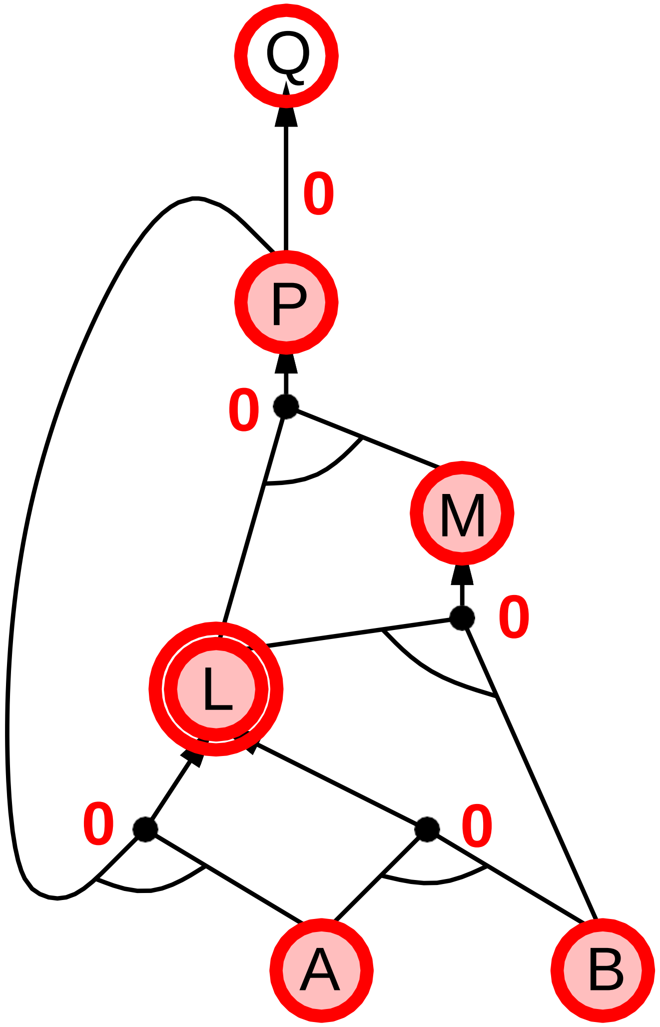


CORK INSTITUTE OF TECHNOLOGY
INSTITIÚID TEICNEOLAÍOCHTA CHORCAÍ

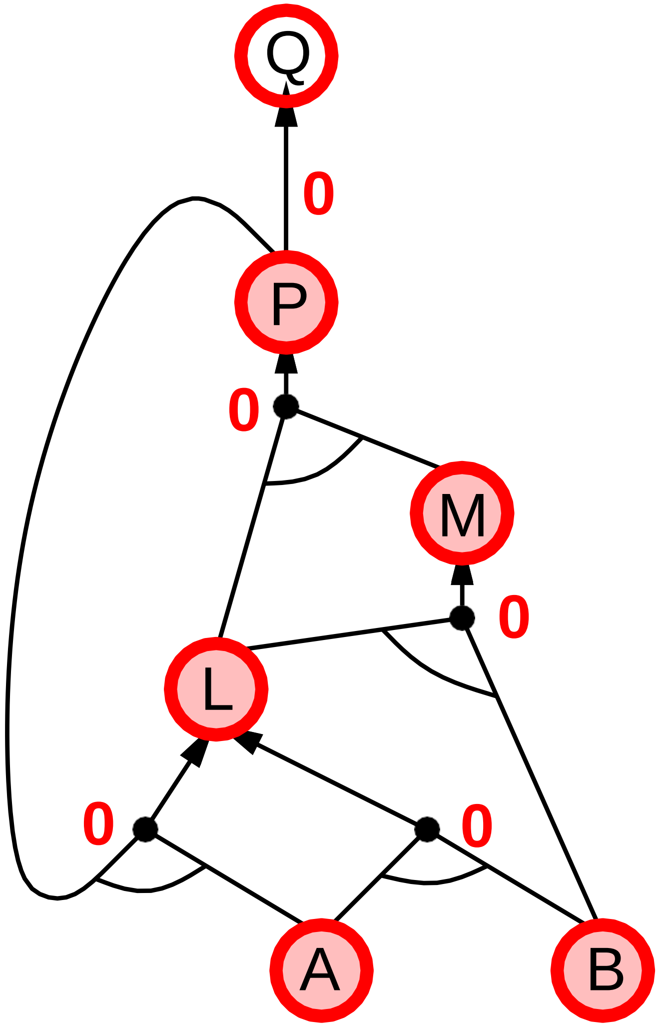
Forward chaining example



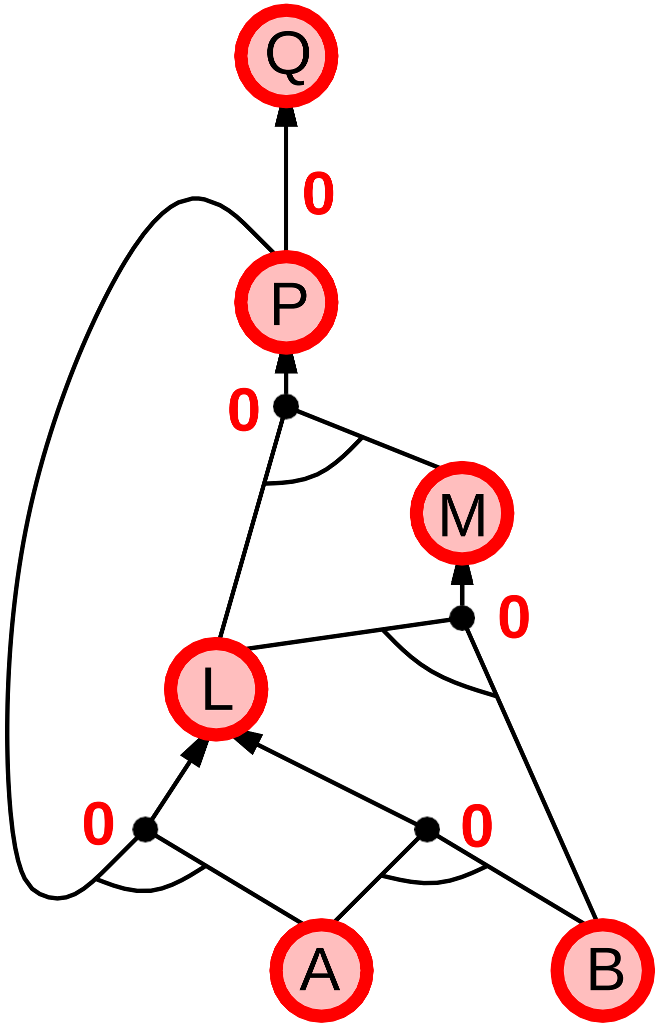
Forward chaining example



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 $a_1 \wedge \ldots \wedge a_k \Rightarrow b$ is **false** in **m**

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

    Therefore the algorithm has not reached a fixed point!

4. Hence ***m*** is a model of ***KB***
5. If *KB* \|= *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*

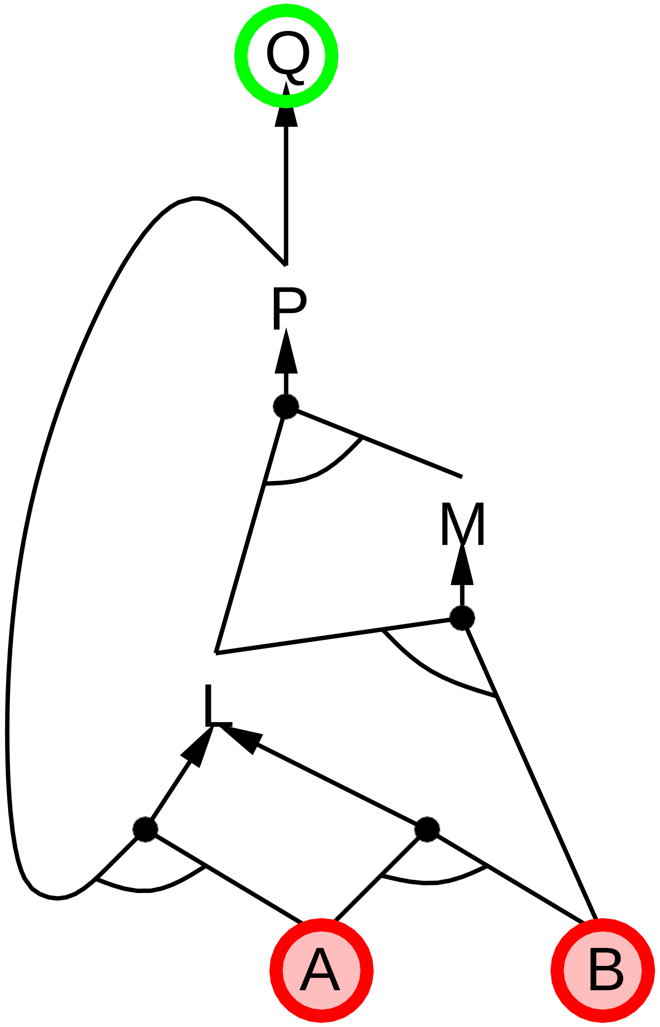
>> Avoid loops: check if new subgoal is already on the goal stack

>> Avoid repeated work: check if new subgoal

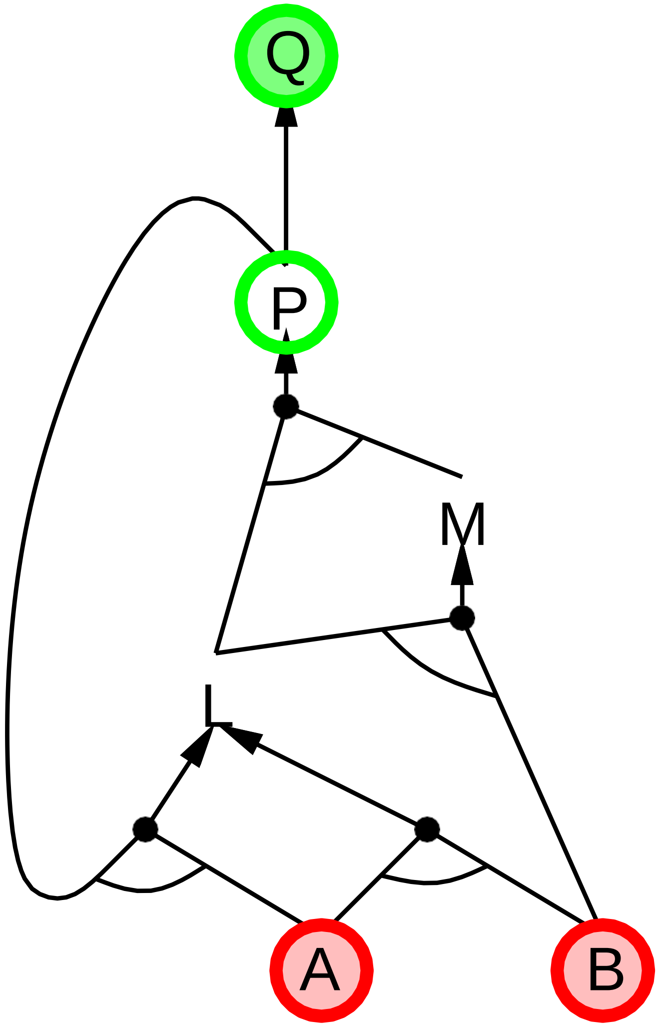
1) has already been proved true, or

2) has already failed

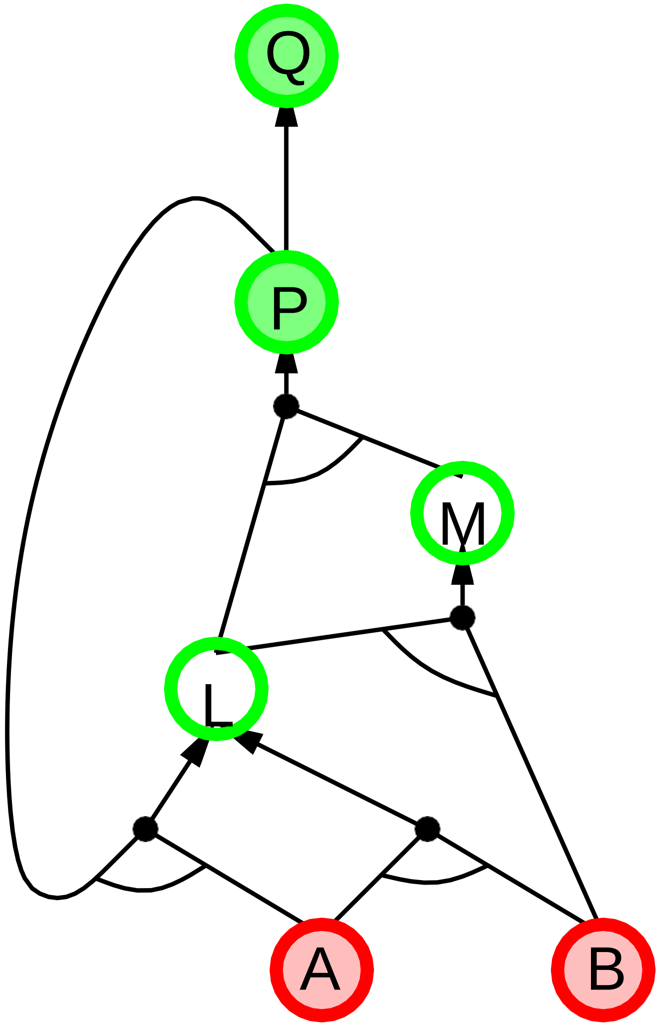
Backward chaining example



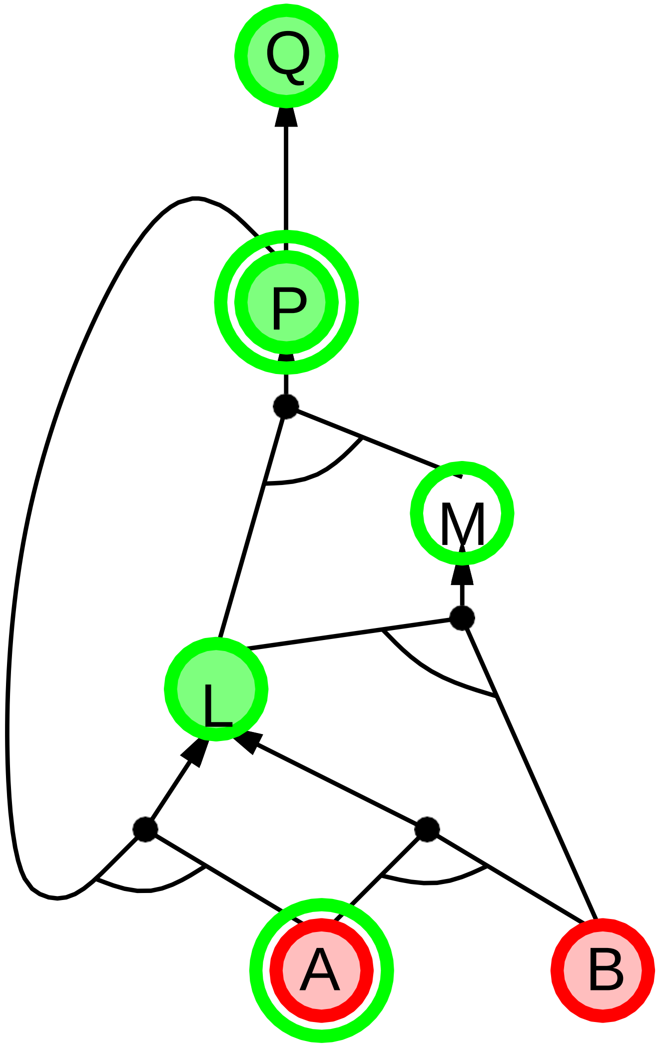
Backward chaining example



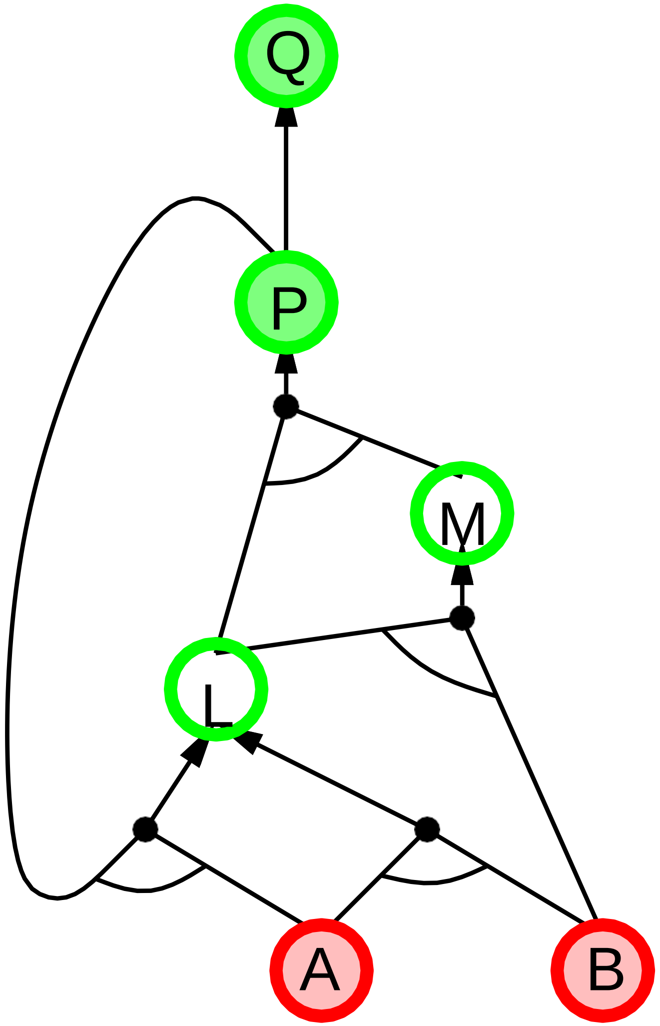
Backward chaining example



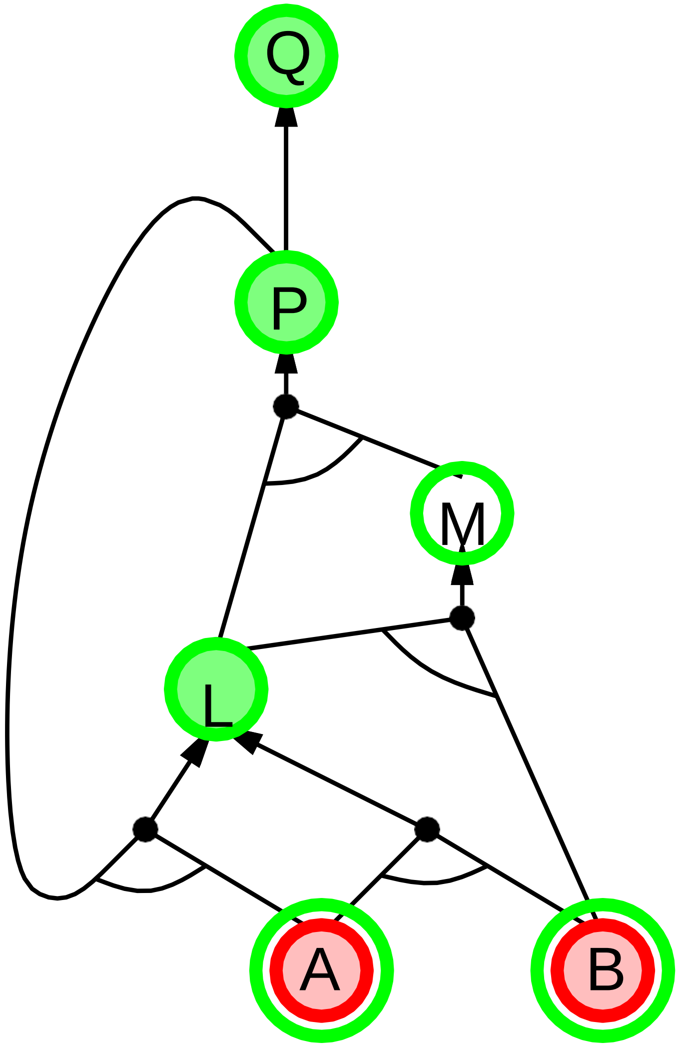
Backward chaining example



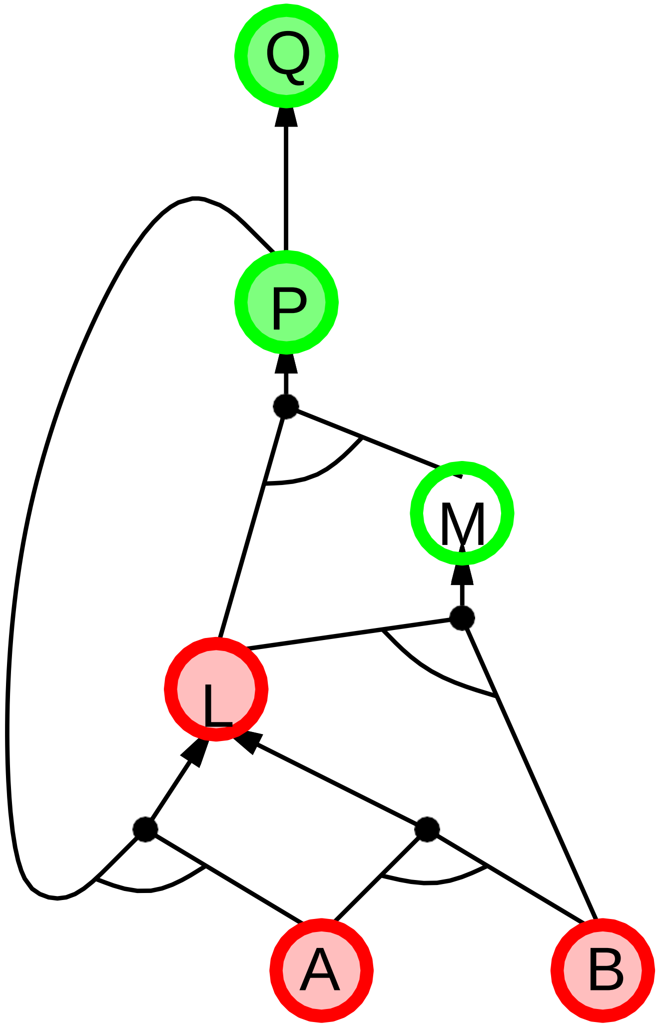
Backward chaining example



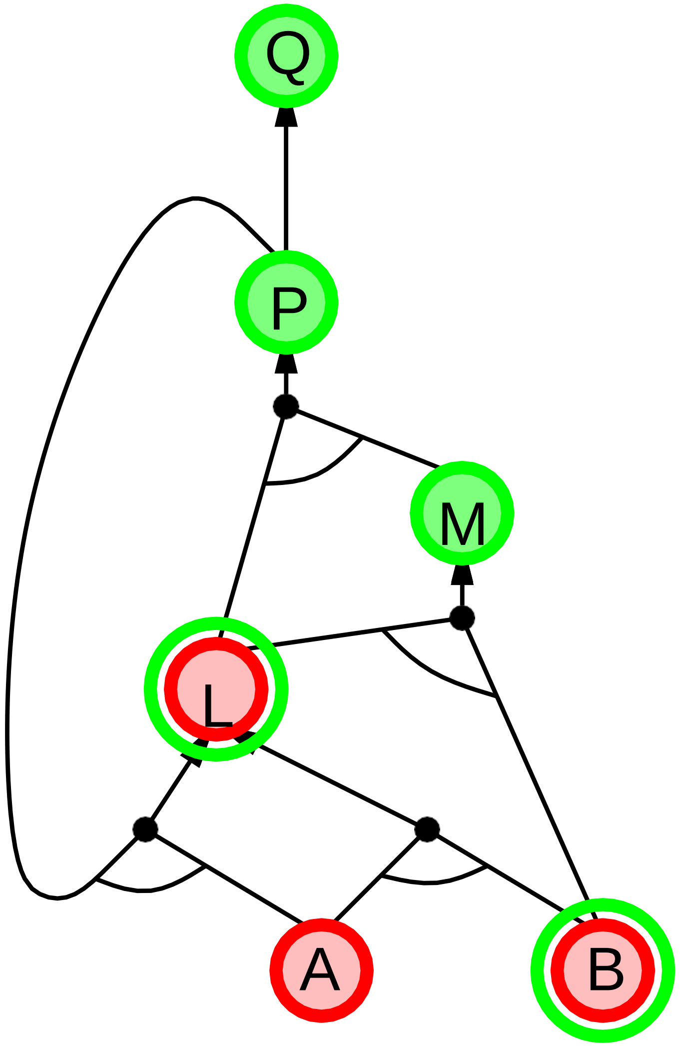
Backward chaining example



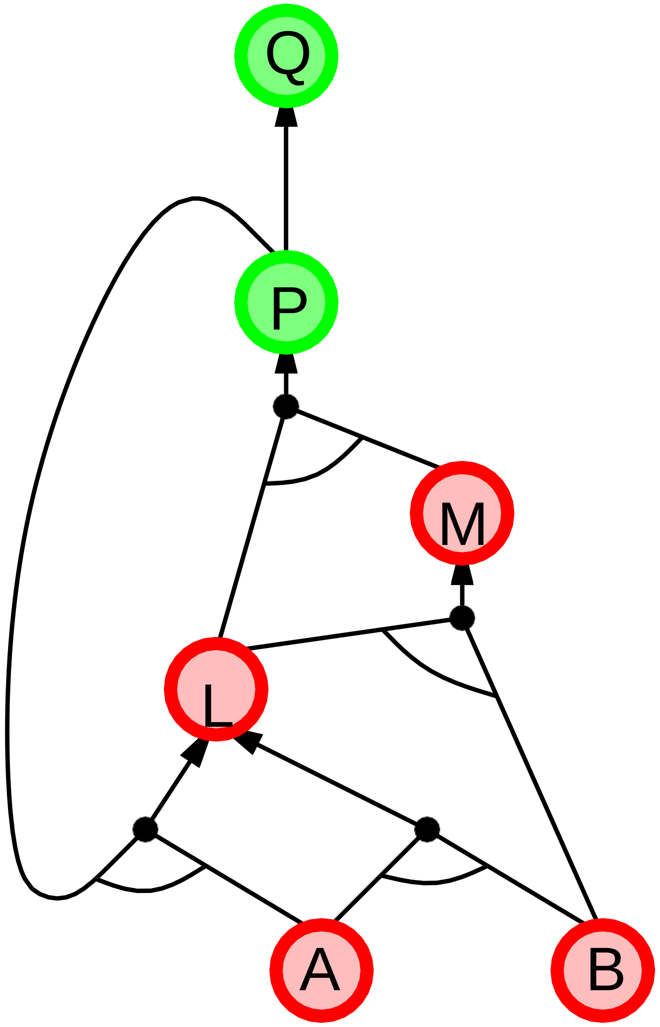
Backward chaining example



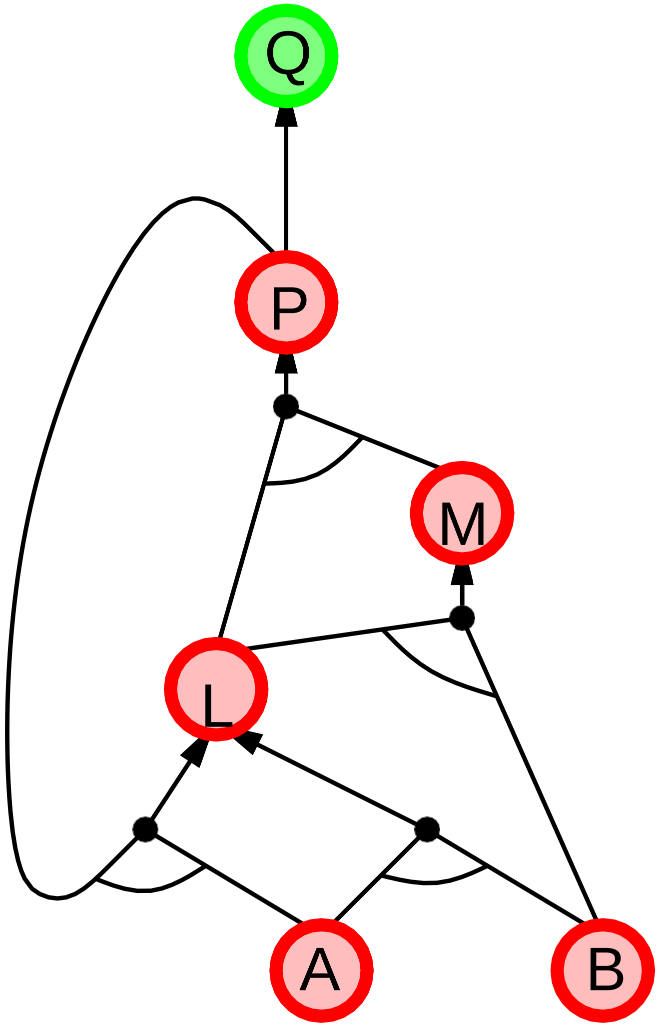
Title



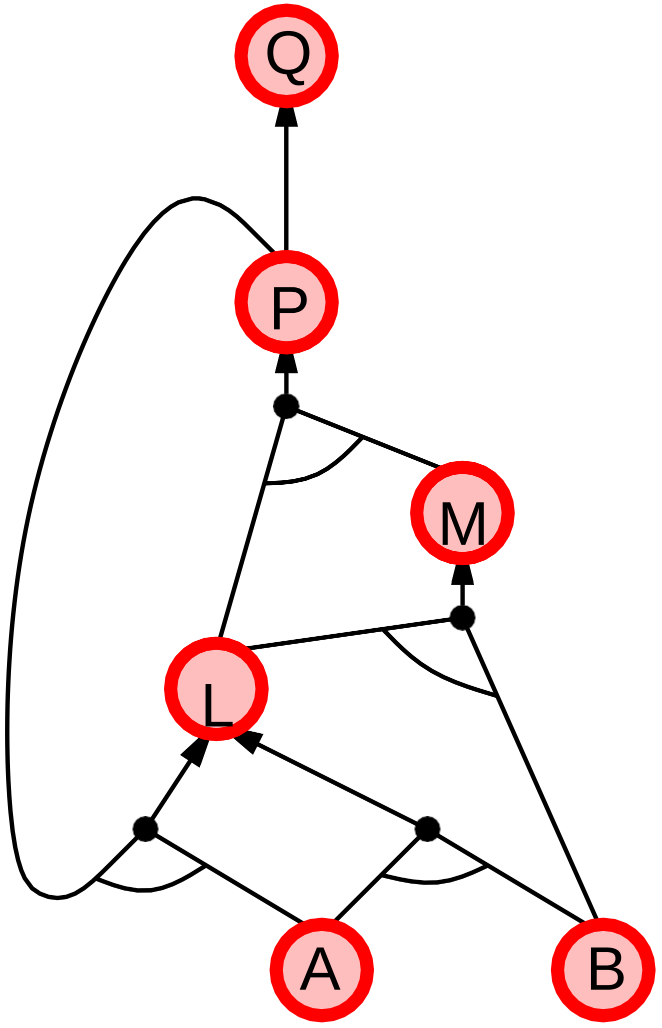
Backward chaining example



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

References

[1] Russell, S. and Norvig, P., 2002. Artificial intelligence: a modern approach Logical Agents, Chapter 7.

[2] – Lecture slides prepared by Russell, S.

Summary