Logical agents

AIMA 3rd ed. - Chapter 7

AIMA 3 - Part II

The problem solving agents studied so far approached the problems with limited knowledge:

- ♢ what their possible actions are
- ♢ resulting states
- ♢ goal state

But they did not know for instance that UP means the reverse of Down (in the N-puzzle problem).

Knowledge-based agents represent the world knowledge (related to their task) in a **knowledge base** and can make **inference** over this knowledge base.

A simple knowledge-based agent

The agent must be able to:

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

Wumpus World PEAS description

Performance measure

gold +1000, death -1000

-1 per step...

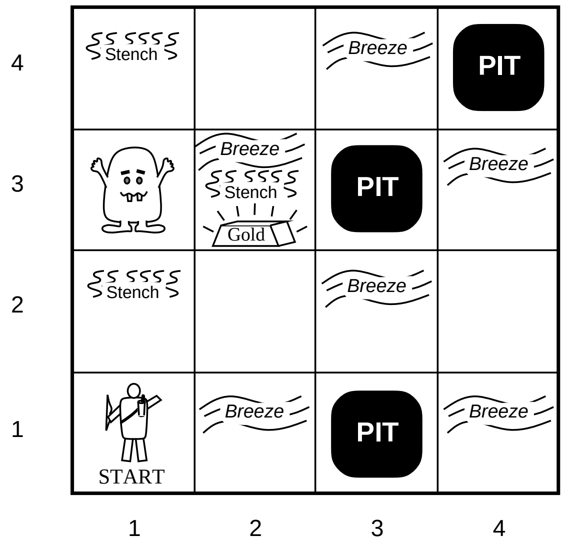
Environment

Squares adjacent to wumpus are smelly

Squares adjacent to pit are breezy

Glitter iff gold is in the same square

...



Actions Up, Down, Left, Right,

Grab, Release, Shoot

Sensors Breeze, Smell, Glitter

Goals Get gold back to start without entering pit or wumpus square

Wumpus world characterization

Fully Observable?? No—only local perception

Deterministic?? Yes—outcomes exactly specified

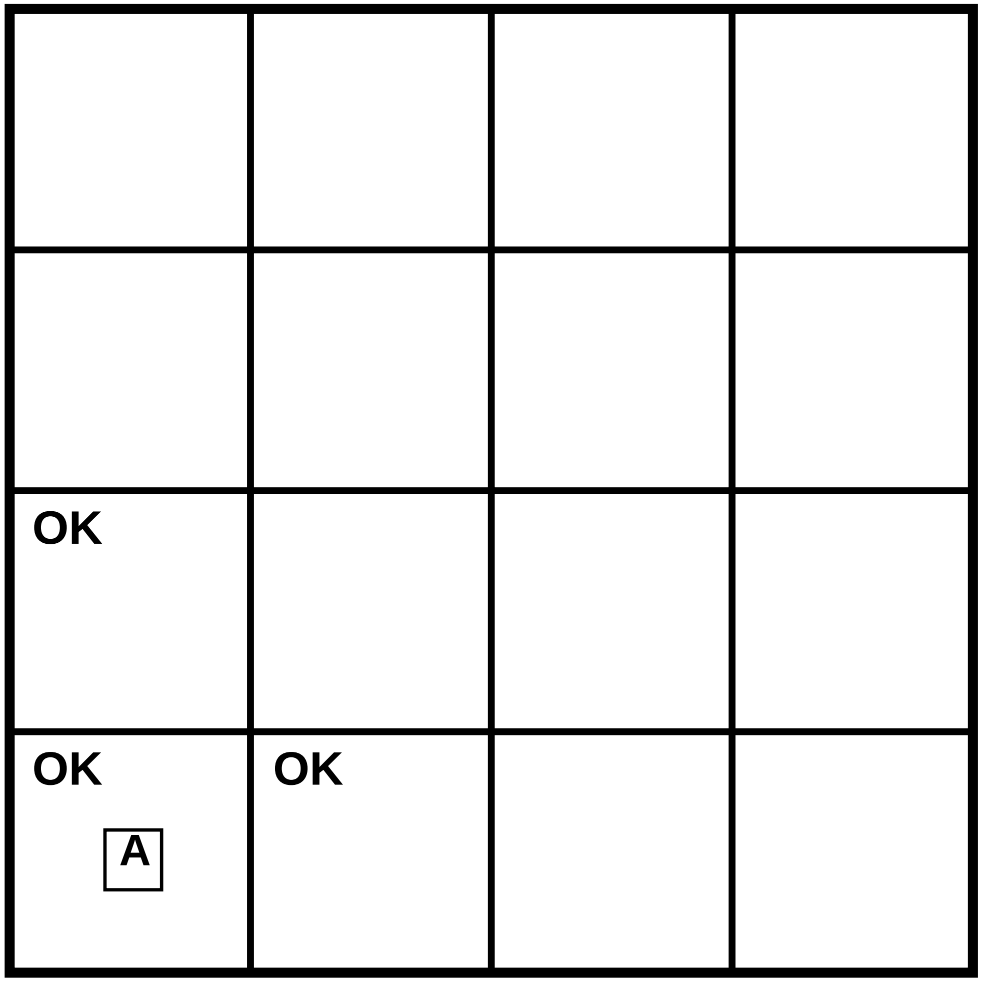
Episodic?? No—sequential at the level of actions

Static?? Yes—Wumpus and Pits do not move

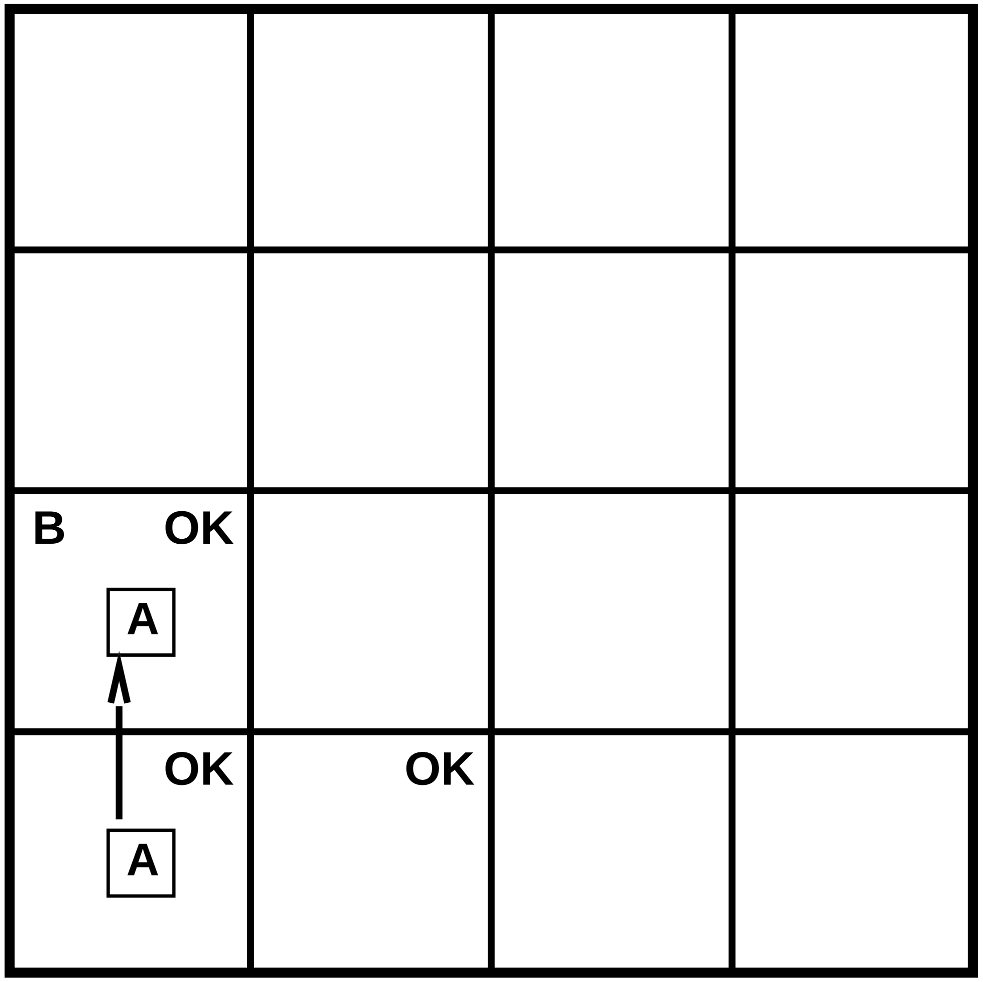
Discrete?? Yes

Single-agent?? Yes—Wumpus is essentially a natural feature

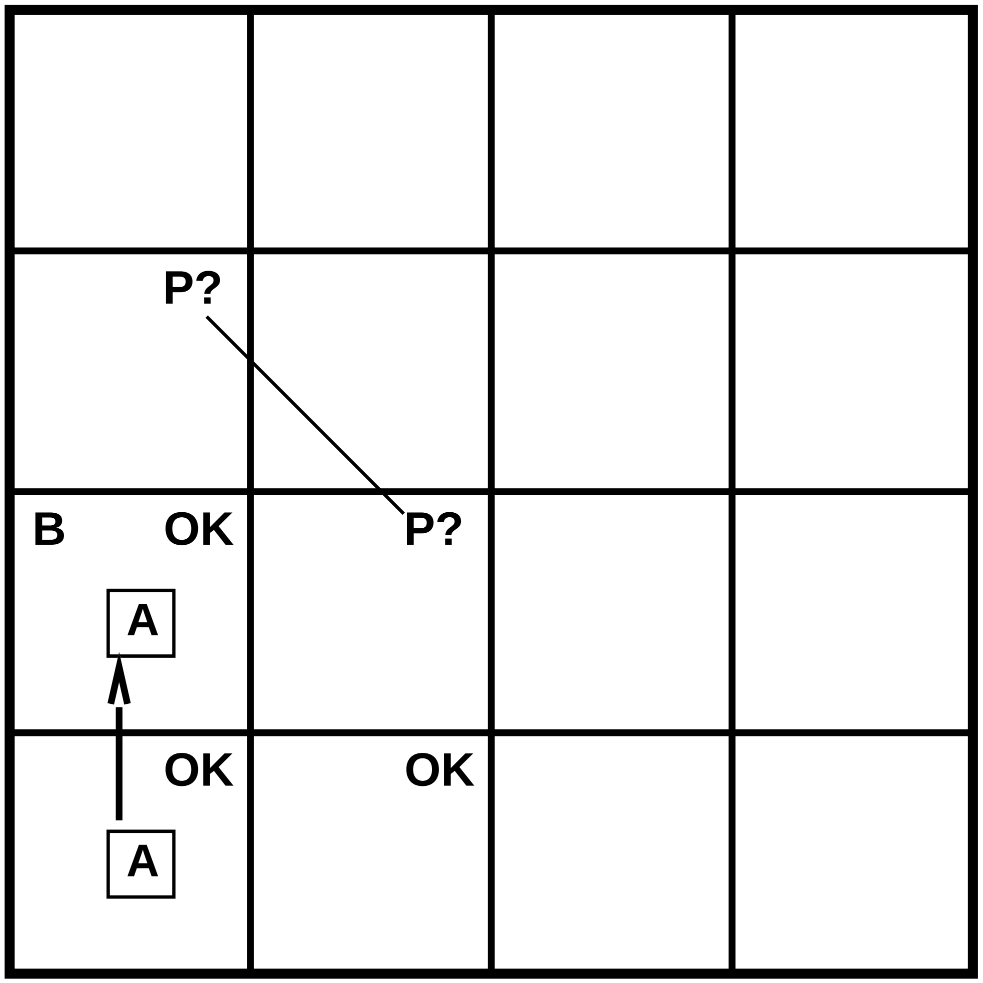
Exploring a wumpus world



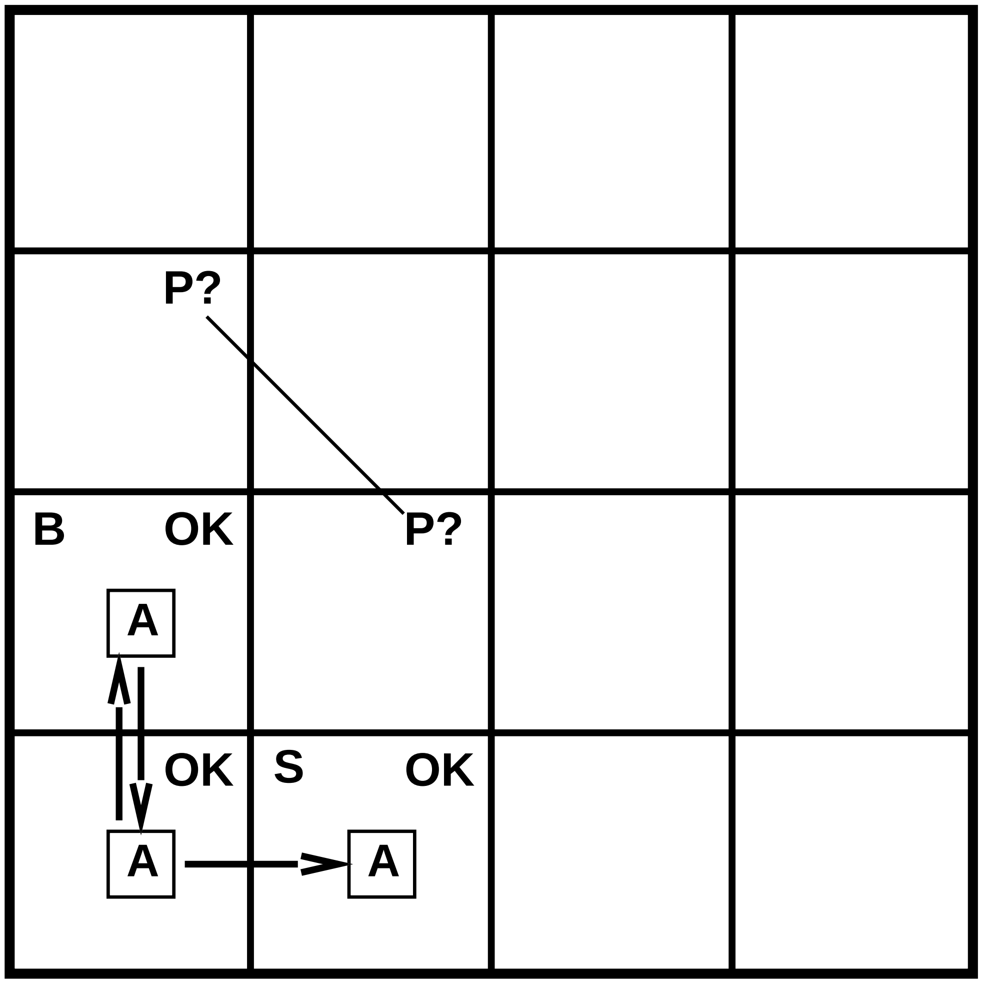
Exploring a wumpus world



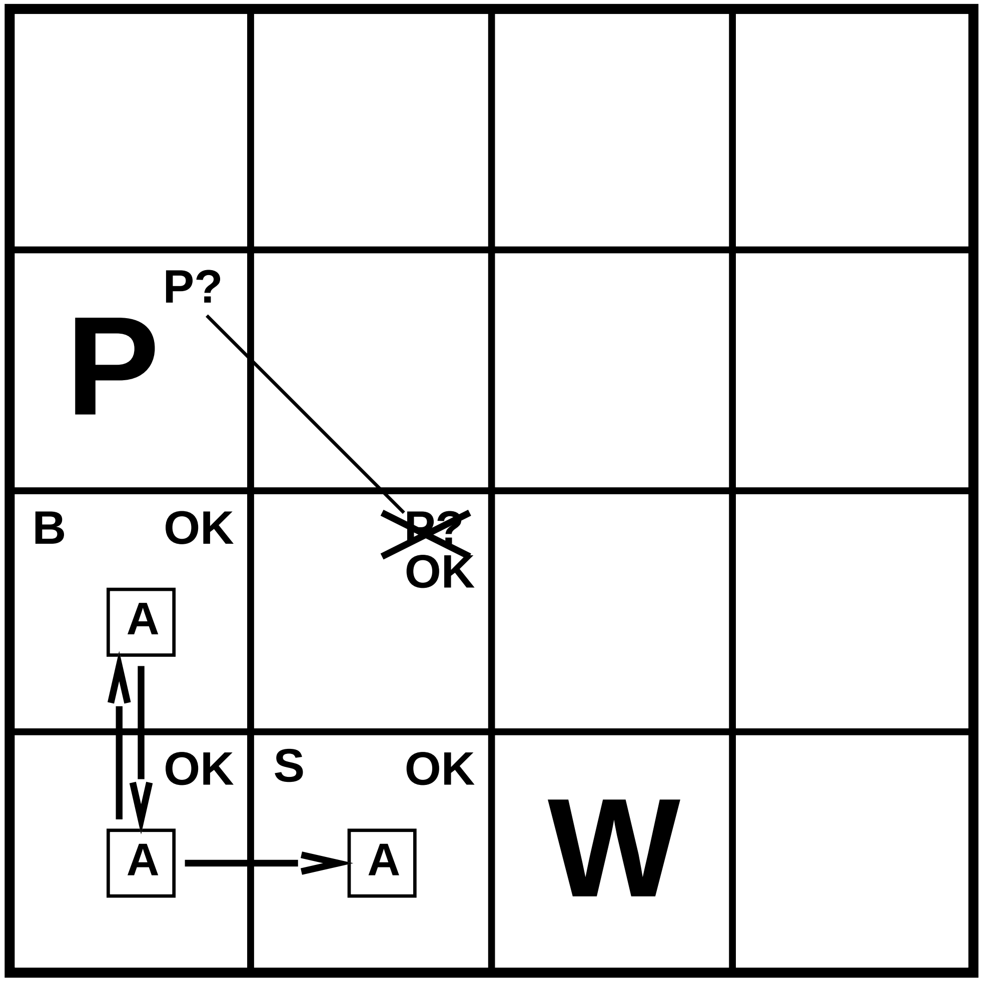
Exploring a wumpus world



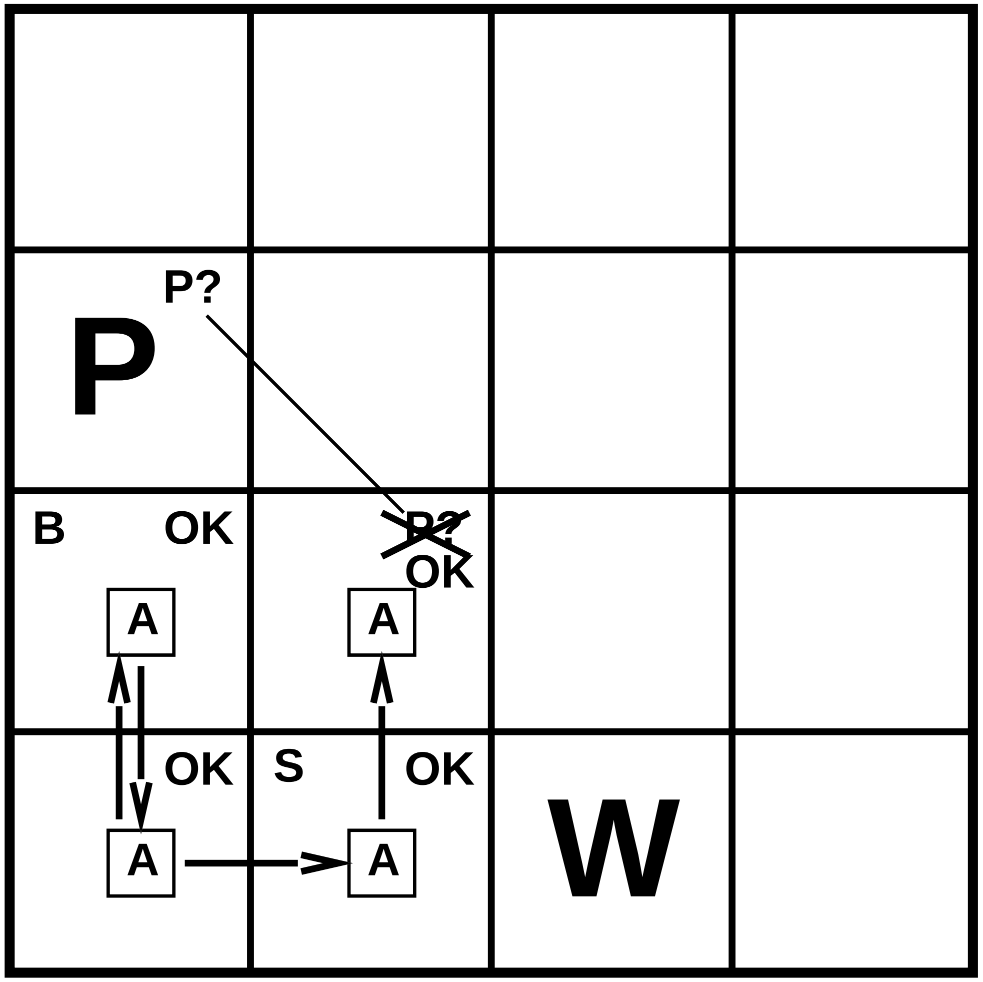
Exploring a wumpus world



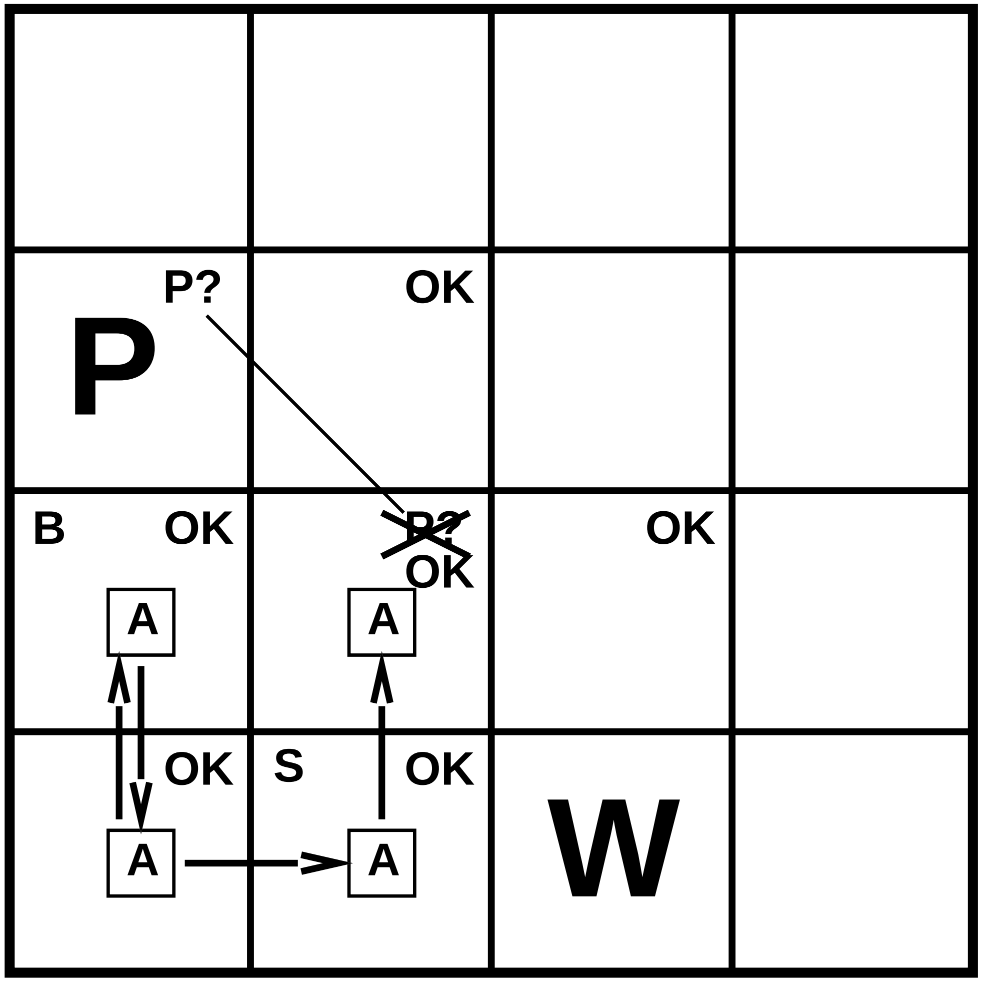
Exploring a wumpus world



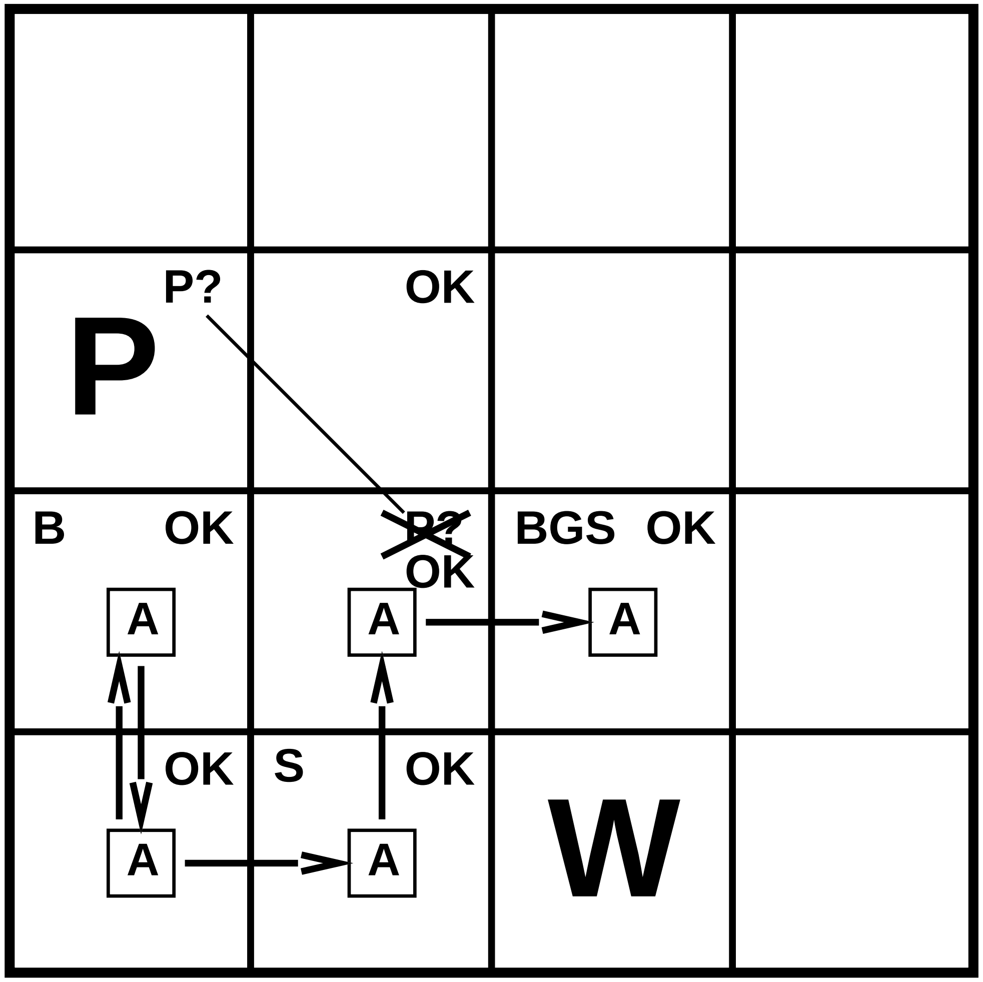
Exploring a wumpus world



Exploring a wumpus world



Exploring a wumpus world



Inference in computers

$\rightarrow$ How can we get an agent do what we did?

That is, how can it know whether it is OK to move to [2,1]?

Notice that an exhaustive search is not an option here because the agent doesnt want to die.

$\rightarrow$ The agent needs a formal inference mechanism to derive valid conclusions

We will use a **formal logic** to extend the capacity of our agents by endowing them with the capacity for **logical reasoning.**

Uncertainty

Note that there may be situations where there are no guaranteed safe moves. In that case, we will consider the **likelihoods of events** (e.g. if we have no choice but go to one of the squares that may have a pit, decide on which one has a higher probability of having a pit; we will see this later).

But for now, we assume that we are dealing with problems that can be solved with logical reasoning and the safety of squares will not be left to chance.

Logic in general

Syntax defines valid sentences in the given language

Semantics define the "meaning" of sentences;
i.e., define truth of each sentence in each possible world

Logic is the framework where a formal language represents information
such that conclusions can be drawn.

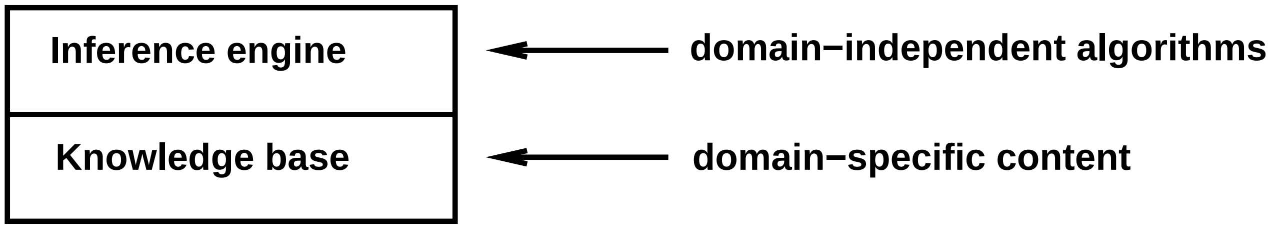
E.g., Language of arithmetic:

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

$x + 2 \geq y$ is true in a world where $x = 7,\ y = 1$
but false in a world where $x = 0,\ y = 6$

Knowledge bases - 6.1

→ Knowledge base is a set of representations of rules and facts (sentences) about the world, expressed in a knowledge representation language.



Knowledge bases

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

TELL it what it needs to know

ASK it what to do—answers should follow from the KB

```
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
```

Knowledge bases

Agents can be viewed at the <u>knowledge level</u>
i.e., what they know, **regardless of how implemented**

The <u>implementation level</u> may represent the knowledge in various forms.
For instance available flights of an airline can be represented as:

- ♢ as a list of strings or
- ♢ as Boolean entries in a 2D table indexed by location pairs etc.

Entailment

$\rightarrow$ We are interested in new sentences that we can infer from the knowledge base and the new percepts.

> For instance, seeing that there is no breeze in [1,1], can we infer that there is no pit in [1,2]?

This can be used when we will "ask" the Knowledge base about suitable actions and to infer hidden properties of the world based on new percepts.

Models

We will first consider **possible worlds** or **models** that define the truth settings of all sentences.

Remember that propositions may have True or False values and we can derive the truth value of more complex sentences based on the propositional logic semantics (slide 34).

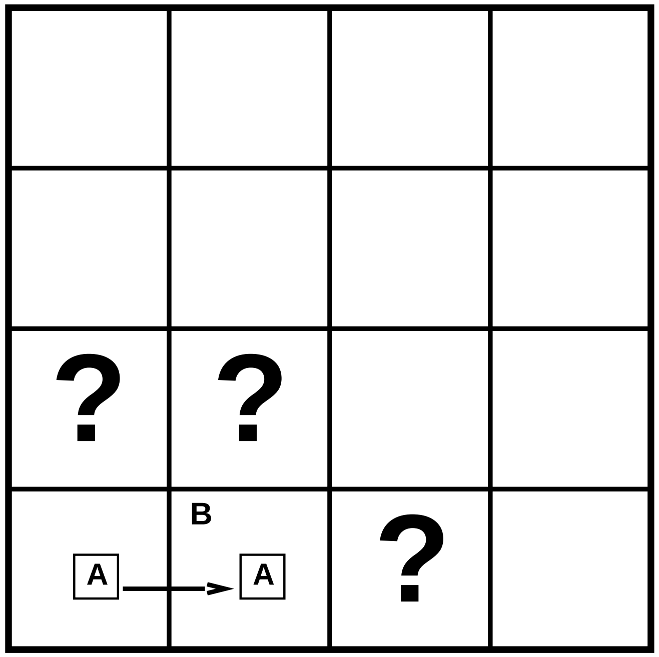
$\rightarrow$ We say m **is a model of a sentence** α **if** α **is true in** m

Note the difference between $world$ and $model$:
"model" is used interchangeably with "world" in the general sense;
but a "model" of a particular sentence is a "world" in which that sentence is True)

$\rightarrow$ We use $M(\alpha)$ to indicate the set of all models of α

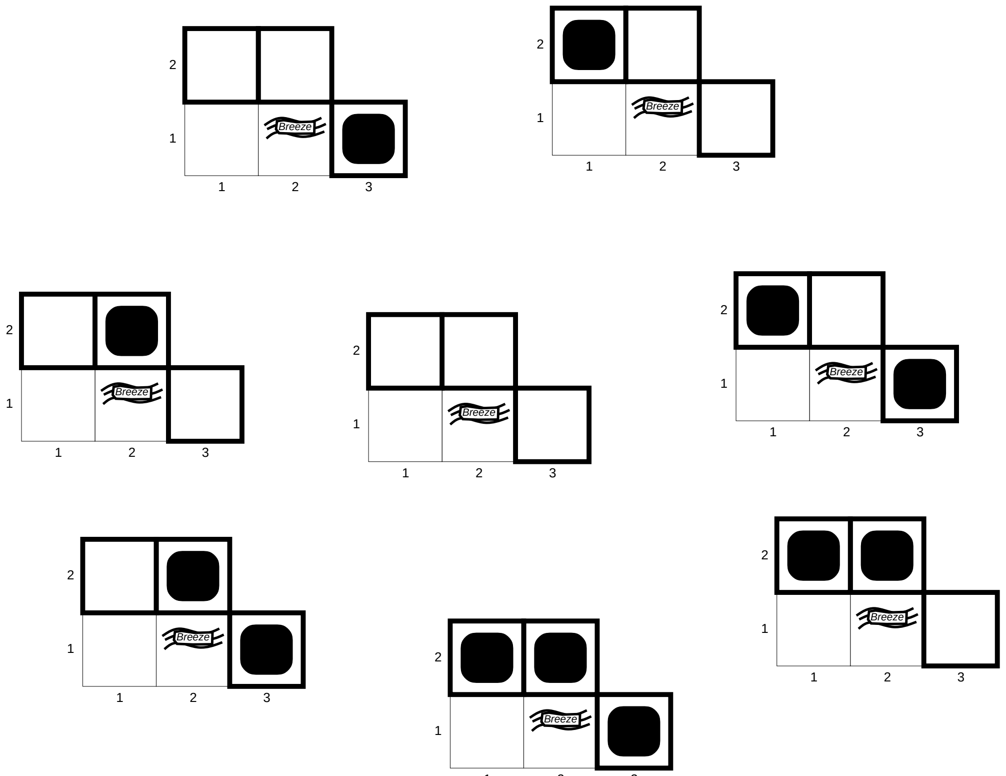
Entailment in the wumpus world

Consider the possible worlds after detecting nothing in [1,1], moving right and detecting breeze in [2,1].



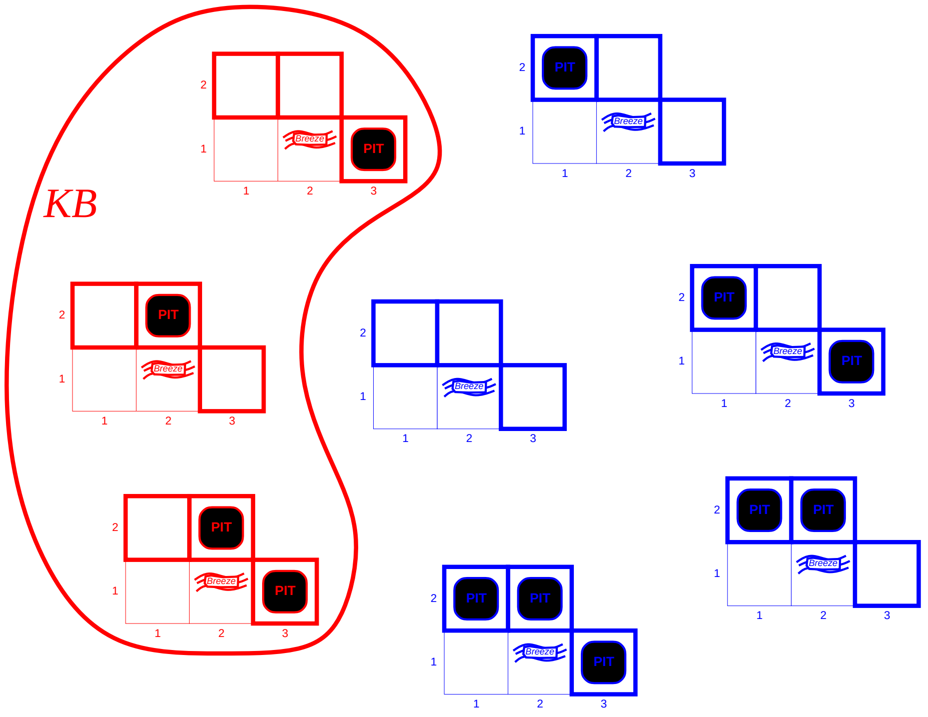
Wumpus models

There are 3 Boolean choices for these three locations, thus 8 possible **worlds** (considering only pits):



Models of KB

There are only 3 worlds that are models of the KB (those worlds are in agreement with the the rules and observations in the KB).



KB = There is no Breeze in 11; there is breeze in 21; and pits cause breeze in adjacent squares

Entailment Definition

Can we deduce that "there is no pit in [1,2]"?

◇ We will use the word **entailment** to mean that one thing follows from another:

E.g., $x = 0$ entails $xy = 0$
E.g., $x + y = 4$ entails $4 = x + y$
E.g., $Fever \wedge Ache$ entails $Fever$

◇ Knowledge base KB entails sentence α (indicated as: $\mathsf{KB} \models \alpha$)
if and only if α is true in all worlds where KB is true

Entailment

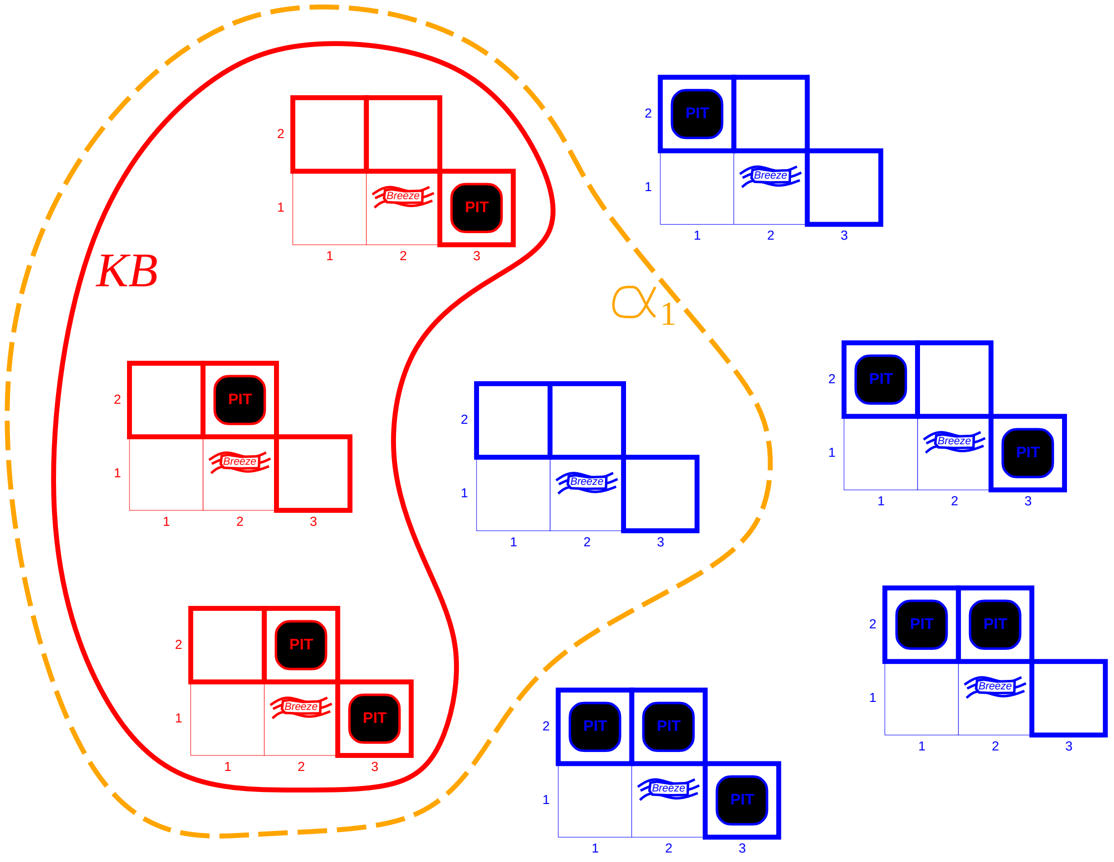
Definition of Entailment:

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

Equivalently;

$KB \models \alpha$ if and only if α is true everywhere KB is true.

Wumpus models

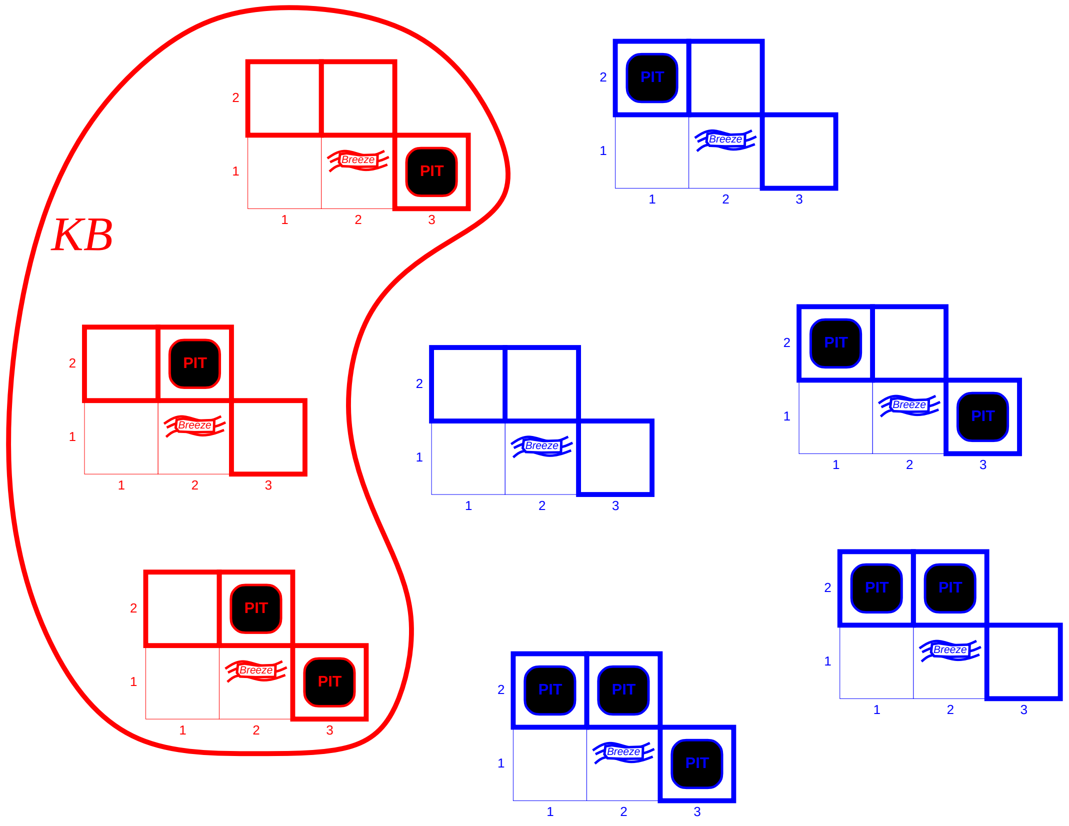


KB = wumpus-world rules + observations

α_1 = "[1,2] is safe", $KB \models \alpha_1$, proved by model checking

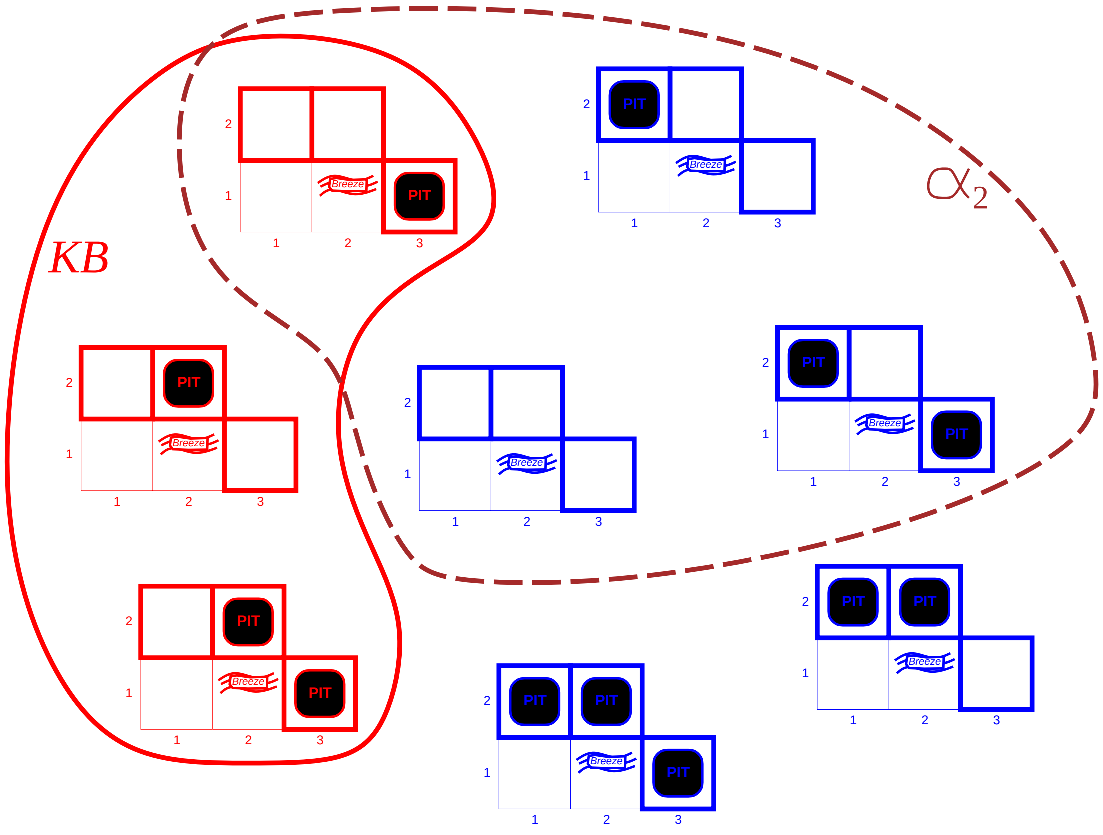
Wumpus models

Does KB entail $\alpha_2 =$ "[2,2] is safe"?



KB = wumpus-world rules + observations

Wumpus models



No, because $M(KB)$ is NOT a subset of $M(\alpha_2)$

Thus, $KB \not\models \alpha_2$

Inference

$KB \vdash_i \alpha$ = 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 $KB \vdash_i \alpha$, 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 use a formal 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.

Representation

Now that we understand entailment, let's see how to represent the facts in the world.

We know of some knowledge representation languages:

◇ Programming languages

In a programming language you can say that "there is a pit in [2,2]" by world[2,2] = pit, but you cannot say "either there is a pit in [2,1] *or* in [3,2]" or that "there is a wumpus in *some* square"

◇ Propositional logic

◇ First-order logic

◇ Natural languages are very **expressive**, but also too flexible/ambiguous

Propositional logic: Syntax

$\rightarrow$ Propositional logic is the simplest logic—illustrates basic ideas

$\rightarrow$ Each proposition symbol P_1, P_2 stands for a proposition that can be true or false and they form the atomic 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

$\rightarrow$ Each model specifies true/false for each proposition symbol:

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

$\rightarrow$ When we will tabulate possible truth value assignments to atomic sentences, each row will constitute a possible world.

The knowldge base (everything we know about the world) will be compatible with some of the rows only (as we have seen in previous slides)

Propositional logic: Semantics

$\rightarrow$ The semantics specifies how to compute the rules for evaluating truth with respect to a model m:

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

Truth tables for connectives

P	Q	$\neg P$	$P \wedge Q$	$P \vee Q$	$P \Rightarrow Q$	$P \Leftrightarrow Q$
false	*false*	*true*	*false*	*false*	*true*	*true*
false	*true*	*true*	*false*	*true*	*true*	*false*
true	*false*	*false*	*false*	*true*	*false*	*false*
true	*true*	*false*	*true*	*true*	*true*	*true*

♢ True is true in every model

♢ False is false in every model

♢ Simple recursive process evaluates an arbitrarily complex sentence, e.g.,
$\neg P_{1,2} \wedge (P_{2,2} \vee P_{3,1}) = false \wedge (false \vee true) = false \wedge true = false$

Truth tables for connectives - Example

$\rightarrow$ Lets fill in the Truth table for the Wumpus world:

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

"There is no pit in $[1, 1]$"
"There is no breeze in $[1, 1]$"
"There is breeze in $[2, 1]$"

$$\neg P_{1,1}$$
$$\neg B_{1,1}$$
$$B_{2,1}$$

Wumpus world sentences

"There is no pit in $[1,1]$"
"There is no breeze in $[1,1]$"
"There is breeze in $[2,1]$"

$$\neg P_{1,1}$$
$$\neg B_{1,1}$$
$$B_{2,1}$$

"Pits cause breezes in adjacent squares"

$$\begin{aligned} P_{1,1} &\Rightarrow (B_{1,2} \wedge B_{2,1}) \\ P_{2,2} &\Rightarrow (B_{1,2} \wedge B_{2,1}) \end{aligned}$$

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

$$\begin{aligned} B_{1,1} &\Leftrightarrow (P_{1,2} \vee P_{2,1}) \\ B_{2,1} &\Leftrightarrow (P_{1,1} \vee P_{2,2} \vee P_{3,1}) \end{aligned}$$

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 true too

R1: $\neg P_{1,1}$

R2: $B_{1,1} \Leftrightarrow (P_{1,2} \vee P_{2,1})$

R3: $B_{2,1} \Leftrightarrow (P_{1,1} \vee P_{2,2} \vee P_{3,1})$

R4 : $\neg B_{1,1}$

R5 $= B_{2,1}$

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 ← 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
        $P$ ← FIRST(symbols); rest ← 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**

Proof methods

Proof methods divide into (roughly) two kinds:

Model checking

a) truth table enumeration (always exponential in n)

b) heuristic search in model space (sound but incomplete)

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

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

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 $\vee$
$\neg(\neg\alpha)$	$\equiv$	α	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** 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:

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

Satisfiability

A sentence is satisfiable if it is true in **some** model

e.g., $A \lor B$, C

A sentence is unsatisfiable if it is true in **no** models

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

Satisfiability is connected to inference via the following:

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

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

Propositional inference: Enumeration method

Let $KB = (A \vee C) \wedge (B \vee \neg C)$ and $\alpha = A \vee B$

Is it the case that $KB \models \alpha$?
Check all possible models—α must be true wherever KB is true

A	B	C	$A \vee C$	$B \vee \neg C$	KB	α
$False$	$False$	$False$				
$False$	$False$	$True$				
$False$	$True$	$False$				
$False$	$True$	$True$				
$True$	$False$	$False$				
$True$	$False$	$True$				
$True$	$True$	$False$				
$True$	$True$	$True$				

Propositional inference: Enumeration method

Let $\alpha = A \vee B$ and $KB = (A \vee C) \wedge (B \vee \neg C)$

Is it the case that $KB \models \alpha$?
Check all possible models: α **must be true wherever** KB **is true**

A	B	C	$A \vee C$	$B \vee \neg C$	KB	α
$False$	$False$	$False$	$False$	$True$	$False$	$False$
$False$	$False$	$True$	$True$	$False$	$False$	$False$
$False$	$True$	$False$	$False$	$True$	$False$	$True$
$False$	$True$	$True$	$True$	$True$	$True$	$True$
$True$	$False$	$False$	$True$	$True$	$True$	$True$
$True$	$False$	$True$	$True$	$False$	$False$	$True$
$True$	$True$	$False$	$True$	$True$	$True$	$True$
$True$	$True$	$True$	$True$	$True$	$True$	$True$

Note that this is equivalent to proving that the sentence
$((A \vee C) \wedge (B \vee \neg C) \;\Rightarrow\; A \vee B$ is a valid sentence.

Complexity of inference

Truth table method: 2^N rows of table for any proof involving N symbols

Linear space requirements though using Depth-First search.

Inference

An inference procedure that generates only entailed sentences is called **sound**

An inference procedure that generates all sentences that can be entailed is called **complete**

Formally:

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

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 $KB \vdash_i \alpha$

Rules of Inference

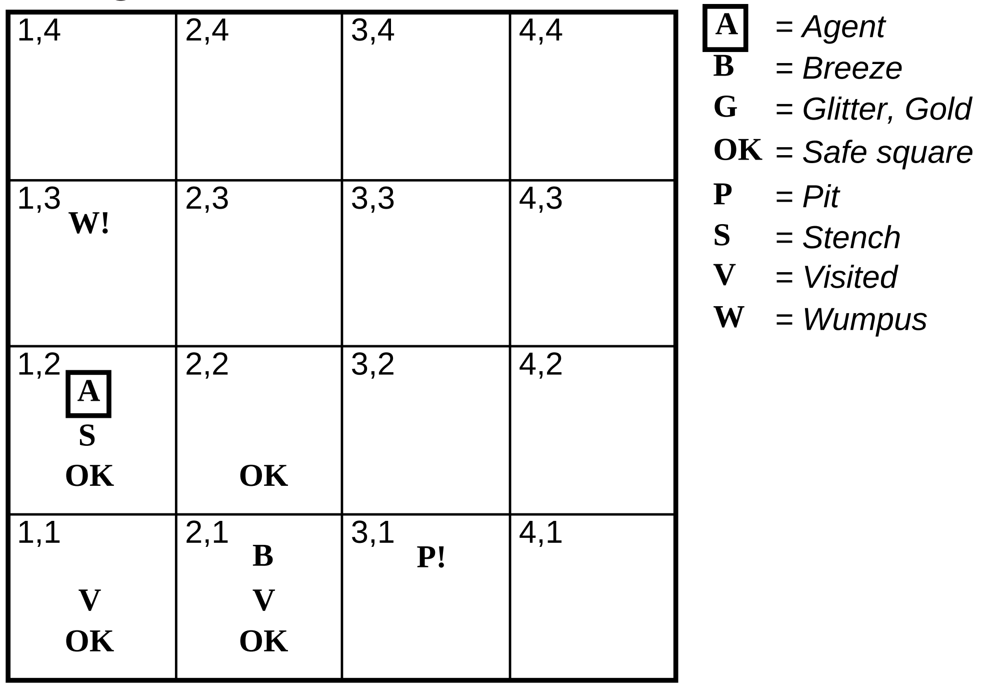
Instead of enumeration method, we can also use inference rules that we know are sound. We list these rules as:

$$\frac{\alpha}{\beta}$$

Notation: Whenever a sentence in the KB matches the pattern above the line (α), the inference rule can conclude the sentence below the line (β).

Application: Propositional inference in the Wumpus world

Assume the following situation:



How does the agent know where to go?

Complete Set of Inference rules for propositional logic

Modus Ponens (or implication elimination)

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

And-elimination (from a conjunction, you can infer any of the conjuncts)

$$\frac{\alpha_1 \wedge \cdots \wedge \alpha_n}{\alpha_i}$$

Unit resolution (from a disjunction, if one of the disjuncts are false, you can infer the other one is true)

$$\frac{\alpha \vee \beta \qquad \neg\beta}{\alpha}$$

Resolution (for CNF): **complete** for propositional logic

$$\frac{\alpha \vee \beta, \qquad \neg\beta \vee \gamma}{\alpha \vee \gamma}$$

Normal forms

Approaches to inference use syntactic operations on sentences which are often expressed in standardized forms:

<u>Disjunctive Normal Form</u> (DNF—**universal***)

disjunction of $\underbrace{\textit{conjunctions} \text{ of } \textit{literals}}_{\textit{terms}}$

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

<u>Conjunctive Normal Form</u> (CNF—**universal***))

conjunction of $\underbrace{\textit{disjunctions} \text{ of } \textit{literals}}_{\textit{clauses}}$

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

* Every sentence can be written in CNF or DNF.

Resolution

♢ Resolution requires the Conjunctive Normal Form.

♢ Resolution is sound and complete for propositional logic

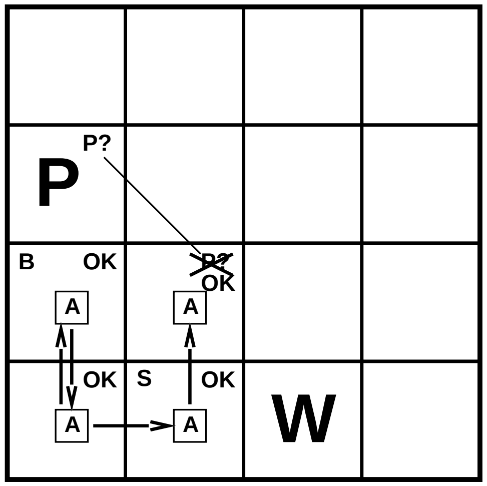
♢ Resolution inference rule (for CNF): complete for propositional logic

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

where ℓ_i and m_j are complementary literals.

♢ E.g.,

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



Resolution algorithm

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

```
function PL-Resolution($KB, \alpha$) returns true or false
    $clauses \leftarrow$ 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

$KB = (B_{1,1} \;\Leftrightarrow\; (P_{1,2} \lor P_{2,1})) \land \neg B_{1,1}$ $\alpha = \neg P_{1,2}$

Add the negation of the sentence you want to prove (e.g. $\neg\neg P_{1,2}$) and see if you reach a contradiction:

Resolution example

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

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:

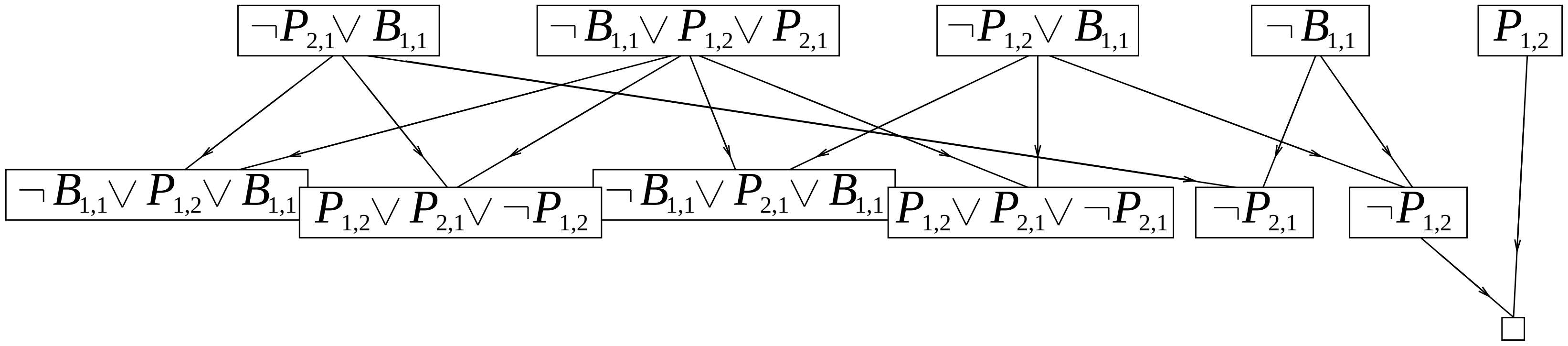
$$(\neg B_{1,1} \vee P_{1,2} \vee P_{2,1}) \wedge ((\neg P_{1,2} \wedge \neg P_{2,1}) \vee B_{1,1})$$

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 example

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



$\neg P_{2,1} \vee B_{1,1}$
$\neg B_{1,1} \vee P_{1,2} \vee P_{2,1}$
$\neg P_{1,2} \vee B_{1,1}$
$\neg B_{1,1}$
$P_{1,2}$
$\neg B_{1,1} \vee P_{1,2} \vee B_{1,1}$
$P_{1,2} \vee P_{2,1} \vee \neg P_{1,2}$
$\neg B_{1,1} \vee P_{2,1} \vee B_{1,1}$
$P_{1,2} \vee P_{2,1} \vee \neg P_{2,1}$
$\neg P_{2,1}$
$\neg P_{1,2}$

Restricted Forms: Horn Form

The completeness of resolution makes it a very important inference method. In many practical situations however, the full power of resolution is not needed.

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

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

Terminology: Body (premise) $\Rightarrow$ Head (conclusion).

Restricted Forms: Horn Form

The Horn form seems very restricted, but many real databases can be written in Horn form. How?

$\diamondsuit$ Definite clauses: disjunctions of literals with exactly one positive literal

e.g.: $(C \wedge D \;\Rightarrow\; B)$

$\diamondsuit$ Facts: a definite clause with no negative literals simply asserts a given proposition

e.g.: C

$\diamondsuit$ Integrity constraints: a Horn clause with no positive literals

e.g.: $\neg(W11 \wedge W22)$

Restricted Forms: Horn Form

How can you convert each sentence into an implication?

$(W11 \land W22) \; \Rightarrow \; False$

Restricted Forms: Horn Form

Modus Ponens is complete for Horn KBs (not in general)

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

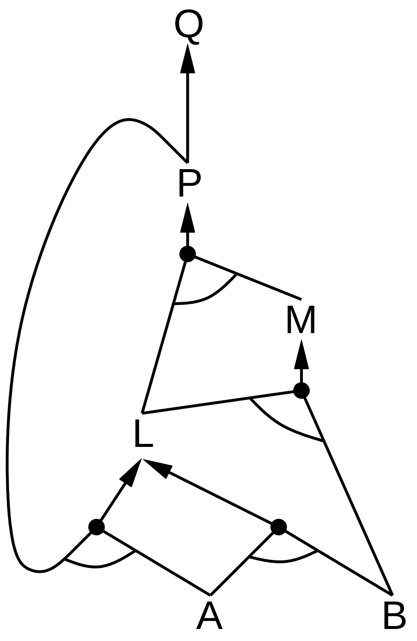
Can be used with forward chaining or backward chaining.
These algorithms are very natural and run in *linear* time in the size of the KB.

They are also sound (easy to show) and complete (slightly more complicated argument).

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 $false$
  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 to be true

  while $agenda$ is not empty do
      $p \leftarrow$ POP($agenda$)
      unless $inferred[p]$ do
          $inferred[p] \leftarrow true$
          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

$P \Rightarrow Q$

$L \wedge M \Rightarrow P$

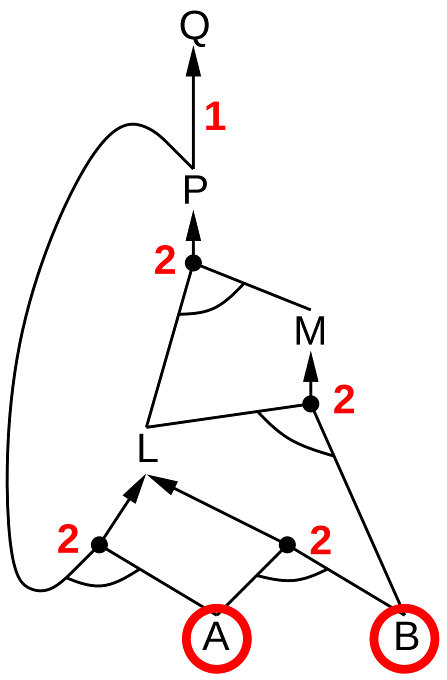
$B \wedge L \Rightarrow M$

$A \wedge P \Rightarrow L$

$A \wedge B \Rightarrow L$

A

B



Forward chaining example

$P \Rightarrow Q$

$L \wedge M \Rightarrow P$

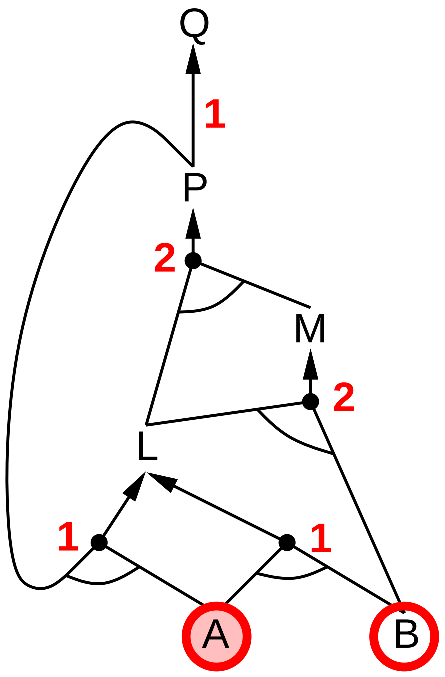
$B \wedge L \Rightarrow M$

$A \wedge P \Rightarrow L$

$A \wedge B \Rightarrow L$

A

B



Forward chaining example

$P \Rightarrow Q$

$L \wedge M \Rightarrow P$

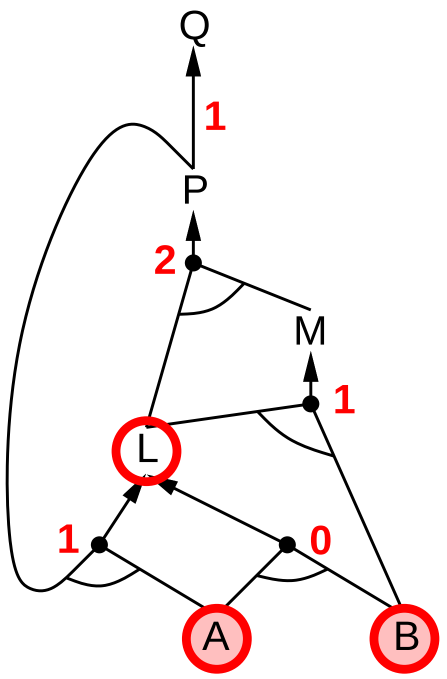
$B \wedge L \Rightarrow M$

$A \wedge P \Rightarrow L$

$A \wedge B \Rightarrow L$

A

B



Forward chaining example

$P \Rightarrow Q$

$L \wedge M \Rightarrow P$

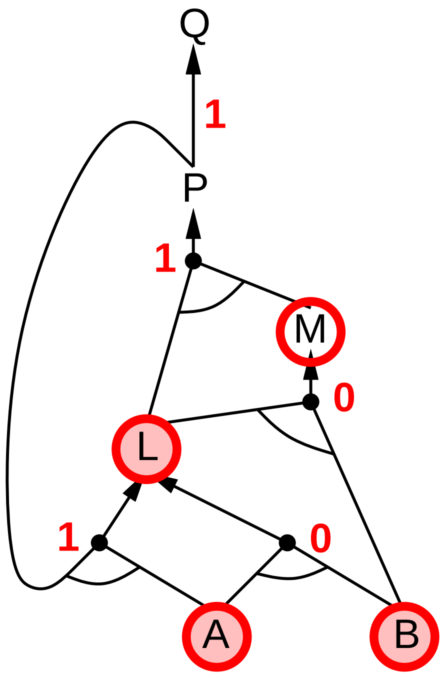
$B \wedge L \Rightarrow M$

$A \wedge P \Rightarrow L$

$A \wedge B \Rightarrow L$

A

B



Forward chaining example

$P \Rightarrow Q$

$L \wedge M \Rightarrow P$

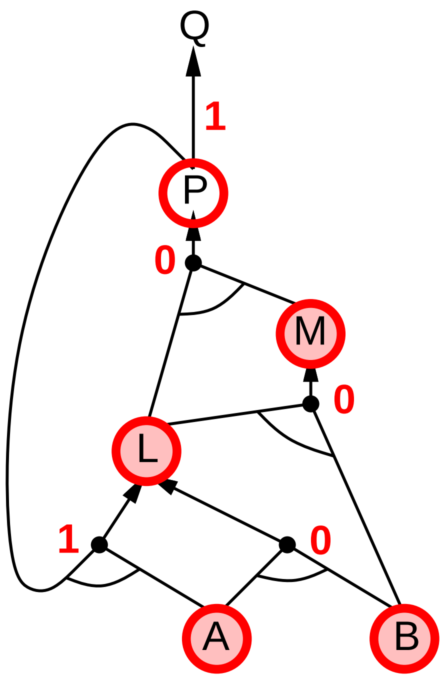
$B \wedge L \Rightarrow M$

$A \wedge P \Rightarrow L$

$A \wedge B \Rightarrow L$

A

B



Forward chaining example

$P \Rightarrow Q$

$L \wedge M \Rightarrow P$

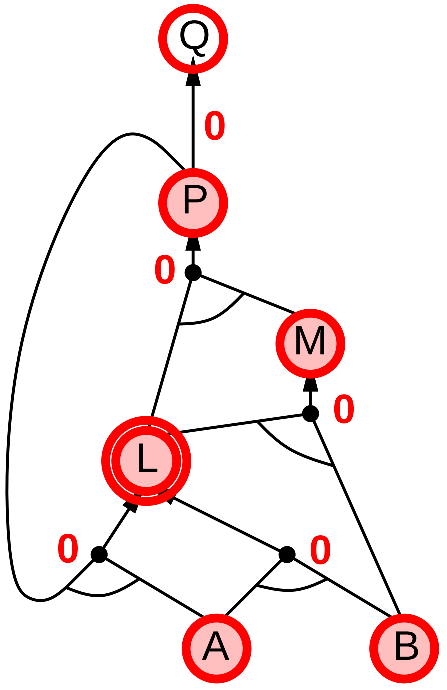
$B \wedge L \Rightarrow M$

$A \wedge P \Rightarrow L$

$A \wedge B \Rightarrow L$

A

B



Forward chaining example

$P \Rightarrow Q$

$L \wedge M \Rightarrow P$

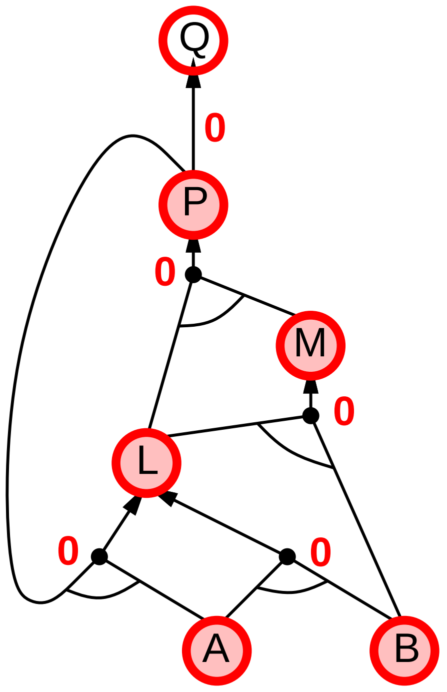
$B \wedge L \Rightarrow M$

$A \wedge P \Rightarrow L$

$A \wedge B \Rightarrow L$

A

B



Forward chaining example

$P \Rightarrow Q$

$L \wedge M \Rightarrow P$

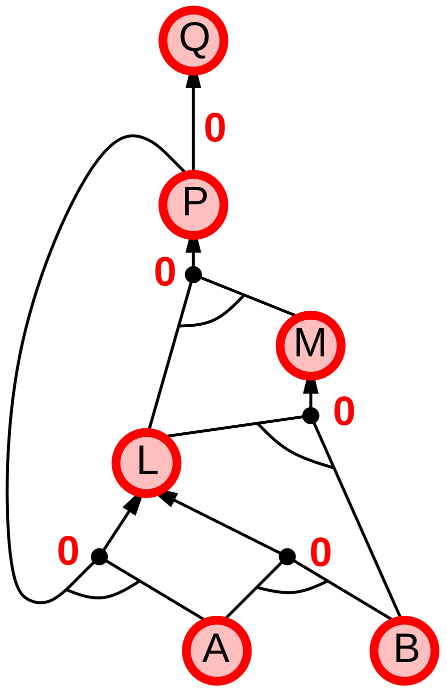
$B \wedge L \Rightarrow M$

$A \wedge P \Rightarrow L$

$A \wedge B \Rightarrow L$

A

B



Backward chaining

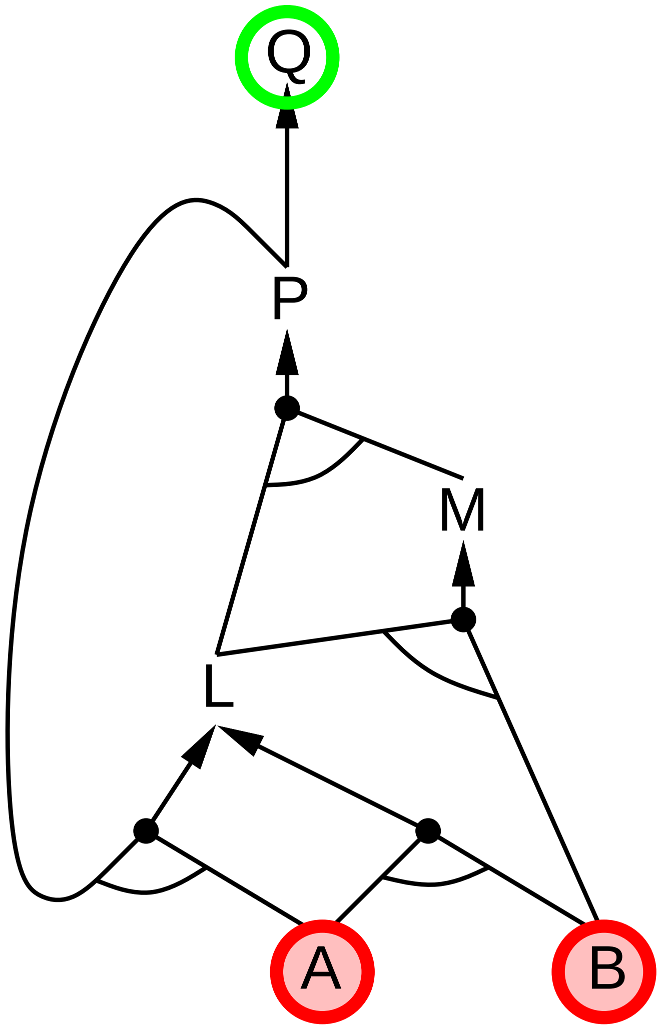
◊ Idea: work backwards from the query q:
to prove q by BC,
check if q is known already, or
prove by BC all premises of some rule concluding q

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

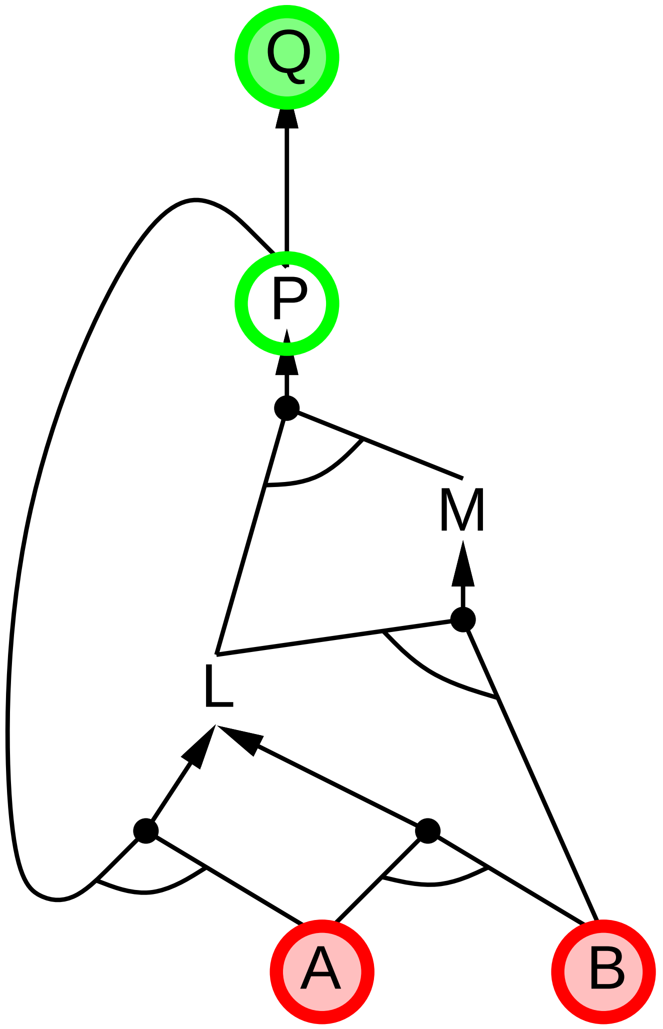
◊ Avoid repeated work: check if new subgoal
1) has already been proved true, or
2) has already failed

◊ The algorithm is essentially identical to the AND-OR-GRAPH search given in Fig. 4.11.

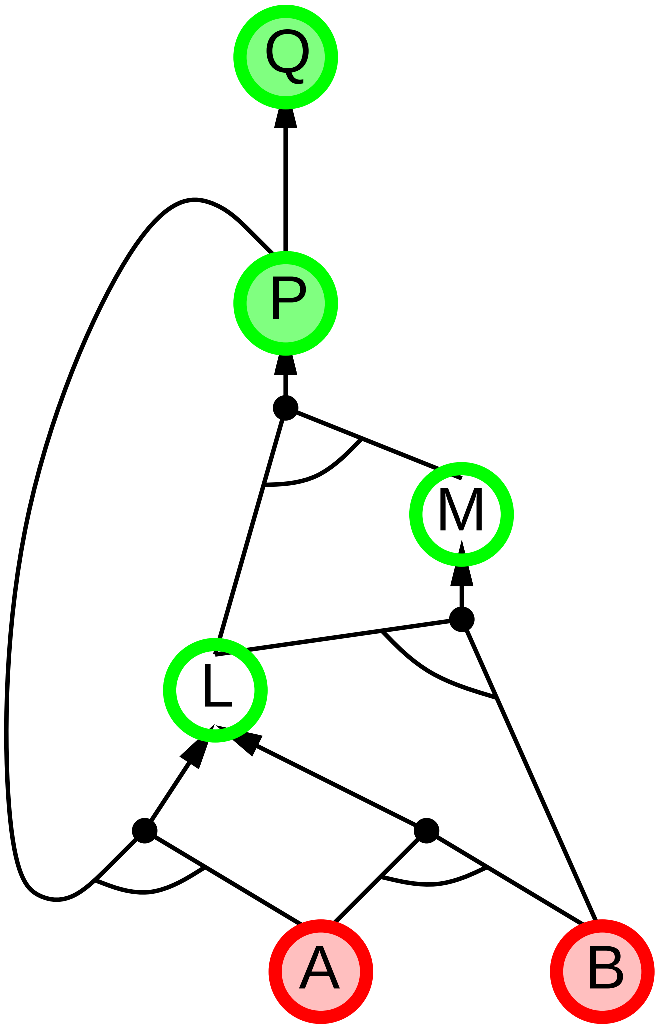
Backward chaining example



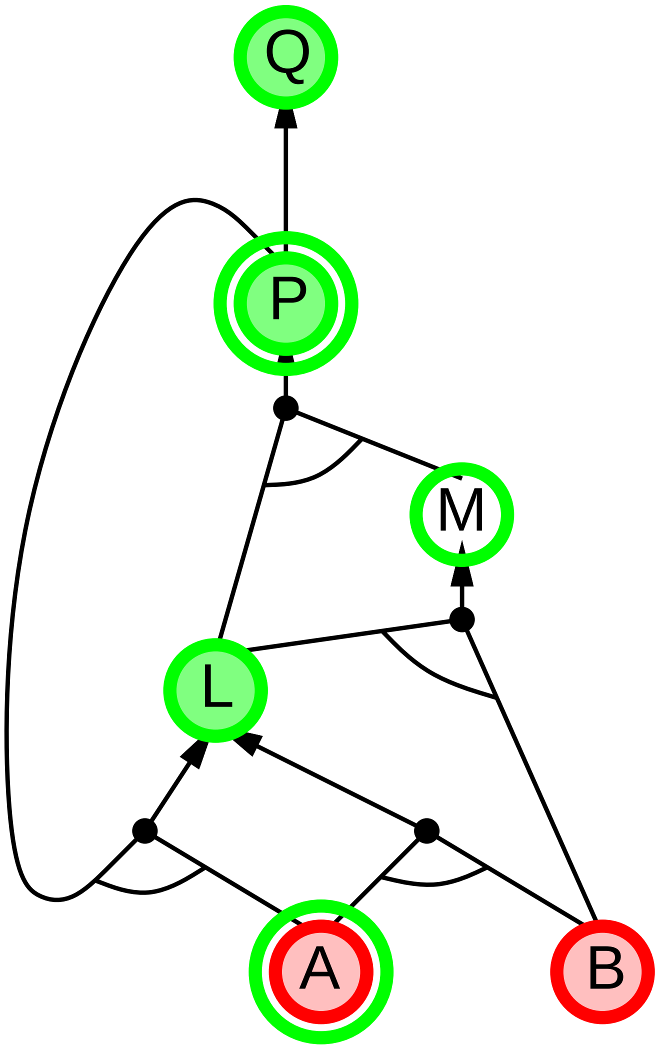
Backward chaining example



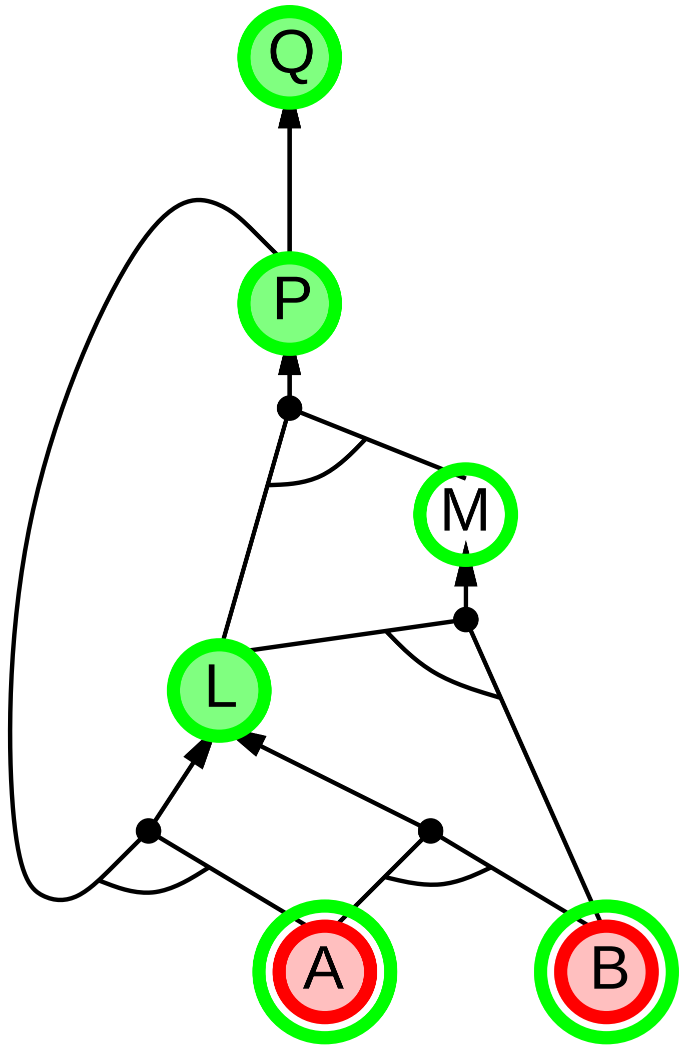
Backward chaining example



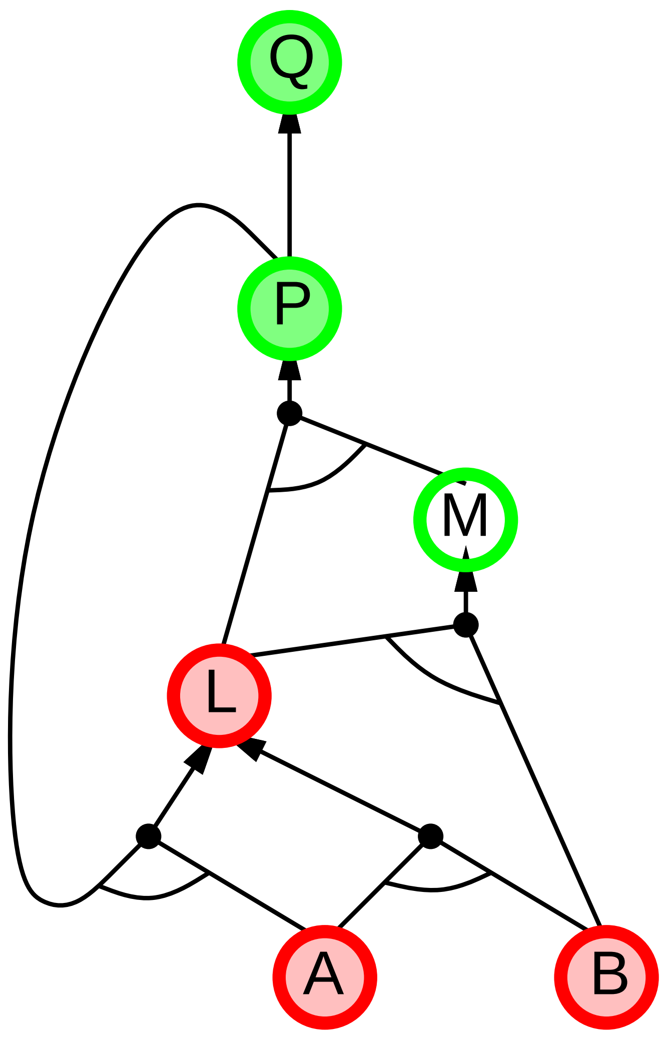
Backward chaining example



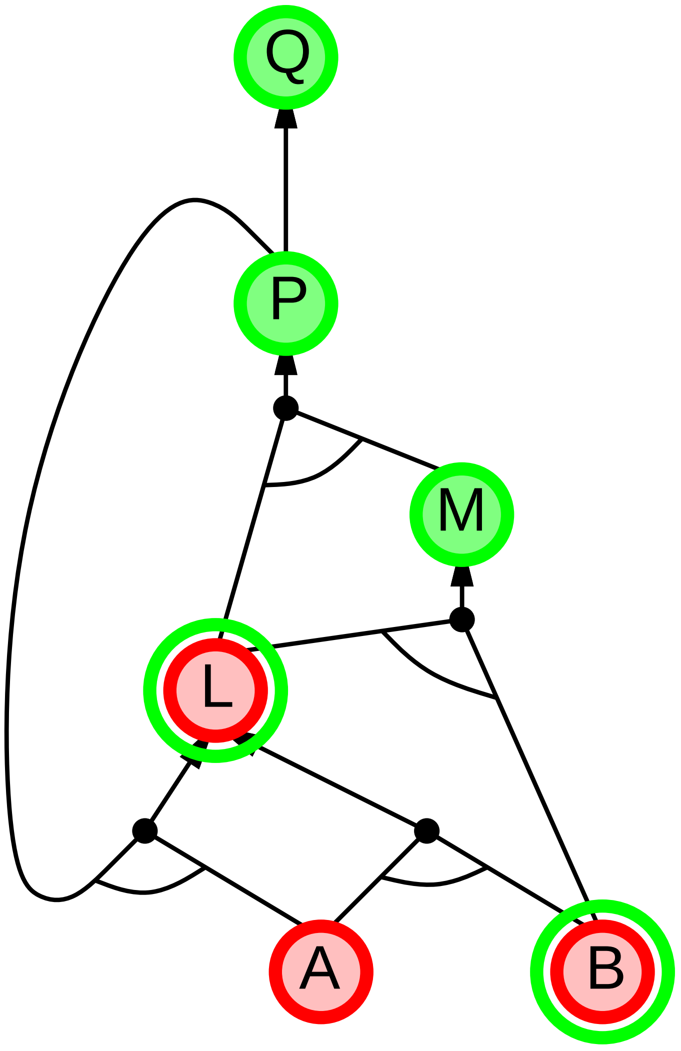
Backward chaining example



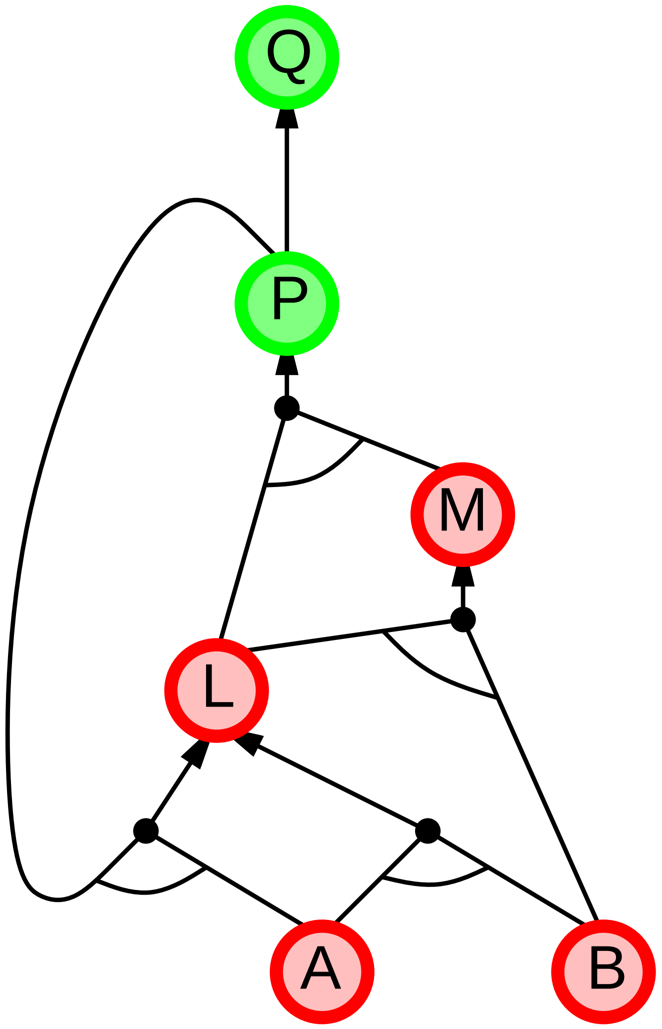
Backward chaining example



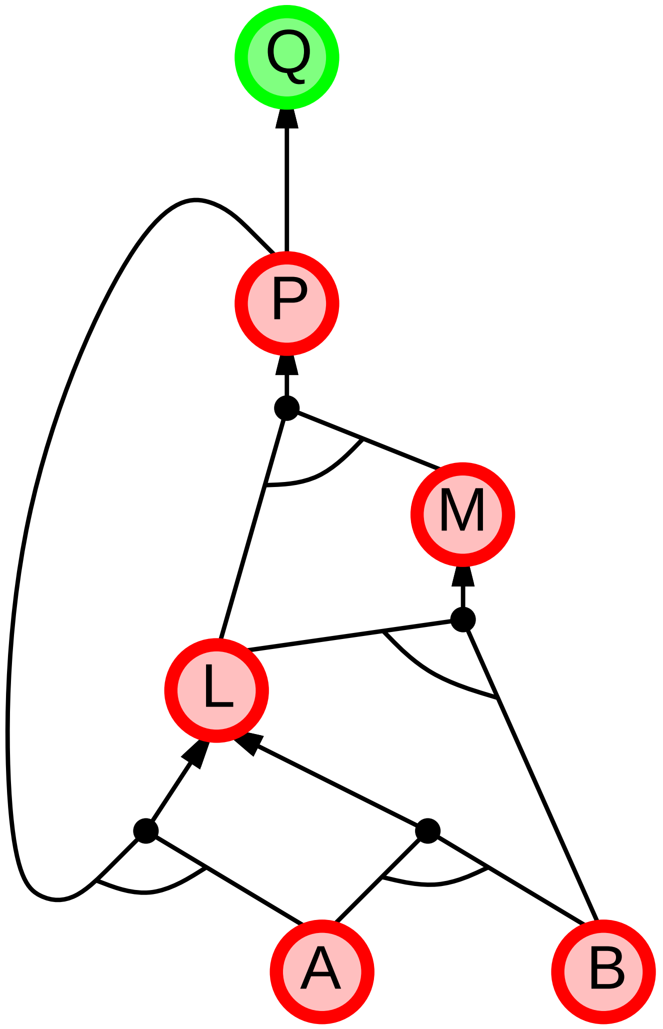
Backward chaining example



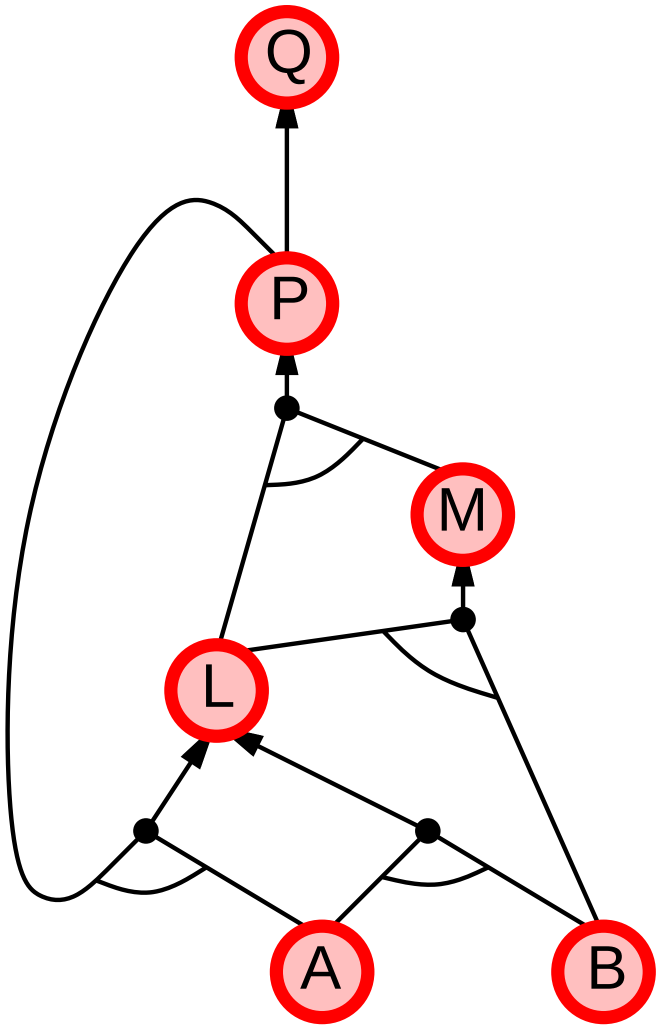
Backward chaining example



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

Satisfiability Problem

Satisfiability problem (is this sentence satisfiable?) is at the core of many general problems (e.g. constraint satisfaction problems).

Satisfiability can be checked by enumerating the possible models until one is found that satisfies the sentence using depth-first enumeration of possible models.

However, determining the satisfiability of sentences in propositional logic was the first problem proved to be NP-complete.

Davis-Putnam (or DPLL) Algorithm

♢ Recursive, depth-first enumeration of possible models,

♢ Similar to TT-ENTAILS but with important improvements such as early termination...

DPLL algorithm is one of the fastest satisfiability algorithms, despite the fact that it is very old. It can solve hardware verification problems with a million variables.

Local search algorithms

We can use local search algorithms for satisfiability problems, as well.

Since each clause in the sentence needs to be satisfied, evaluation function should count the number of unsatisfied clauses (e.g. min-conflict heuristic).

WALKSAT algorithm is one of the most effective algorithms.

WALKSAT algorithm

♢ In each iteration, pick an unsatisfied clause and picks a symbol in it to flip.

♢ Two ways to decide how to choose the symbol:

- randomly select the symbol from the clause and flip it with probability p

- pick one that minimizes the number of unsatisfied clauses in the new state

Local search algorithms

In underconstrained situations, solutions are easy to find.

Ex. A 3-CNF with 5 symbols and 5 clauses:

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

How many solutions?

What makes it a hard satisfiability problem?

What if WALKSAT returns failure?

Local search algorithms

In underconstrained situations, solutions are easy to find.

Ex. A 3-CNF with 5 symbols and 5 clauses:

$$(\neg D \lor \neg B \lor C) \land (B \lor \neg A \lor \neg C) \land (\neg C \lor \neg B \lor E)$$
$$\land (E \lor \neg D \lor B) \land (B \lor E \lor \neg C)$$

How many solutions?

16 of the 32 possible worlds are models for this sentence!

What makes it a hard satisfiability problem?

How constrained it is (w.r.t number of constraints and number of variables)

What if WALKSAT returns failure?

We cannot tell if the sentence is unsatisfiable or just WALKSAT needs more time.

Shortcomings of PL

Lack of Expressive Efficiency:

We need to state the general rule about breeze and pits as a separate sentence for each square:

$B11 \;\Rightarrow\; P21 \vee P12$

$B21 \;\Rightarrow\; P11 \vee P22 \vee P31$

Shortcomings of PL - Fluents

Difficulty Dealing with Fluents (changing aspects of the world) :

E.g. Keeping track of location and orientation.

Let L_{ij} be a location proposition. Then we would like to say something like:

$L_{11} \wedge FacingRight \wedge Forward \;\Rightarrow\; L21$

But then we will have both L_{11} and $L21$ in the database!

Shortcomings of PL - Fluents

Difficulty Dealing with Fluents (changing aspects of the world) :

E.g. Keeping track of location and orientation.

Let L_{ij} be a location proposition. Then we would like to say something like:

$L_{11} \wedge FacingRight \wedge Forward \;\Rightarrow\; L21$

But then we will have both L_{11} and $L21$ in the database!

Solution: Associate propositions with time steps:

$L_{11}^0, FacingEast^0, HaveArrow^0...$

Shortcomings of PL - Fluents

$\rightarrow$ After associating time with prepositions, we can derive properties of squares:

$$L^t_{xy} \Rightarrow (Breeze^t \Leftrightarrow Breeze_{xy})$$

$\rightarrow$Now we also need to keep track of fluents

Solution: Effect axioms: Add axioms that specify the outcome of an action aat the next time step

$$L^0_{11} \land FacingEast^0 \land Forward^0 \Rightarrow (L^1_{21} \land \neg L^1_{11})$$

But we would need one such sentence for each possible time step and each possible square and orientation!

Shortcomings of PL - Frame Problem

$\rightarrow$ The effect axioms fail to state what is unchanged as the result of an action.

E.g. After taking a step forward in the first square, the agent still has an arrow.

Solution: **Frame axioms**: Add axioms that explicitly asserts all the propositions that remain the same:

$$Forward^t \Rightarrow (HaveArrow^t \Leftrightarrow HaveArrow^{t+1})$$

$$Forward^t \Rightarrow (WumpusAlive^t \Leftrightarrow WumpusAlive^{t+1})$$

...

Shortcomings - Representational Frame Problem

$\rightarrow$ The proliferation of frame axioms is very inefficient and this problem was a significant problem for AI researchers.

Solution: **successor-state axioms**: Shift focus from writing axioms about actions to writing axioms about fluents:

"Agent has an arrow if it had an arrow and did not shoot" $HaveArrow^{t+1} \Leftrightarrow (HaveA$ $\neg Shoot^{t})$

"The agent will be in L_{11}^{t+1} at time $t+1$ if it was there and did not/could not move forward, or that it was in L_{12} at time t facing south and moved forward..." $L_{11}^{t+1} \Leftrightarrow (L_{11}^{t} \wedge (\neg Forward^{t} \vee Bump^{t+1}))$
$\vee(L_{12}^{t} \wedge (\neg FacingSouth^{t} \wedge Forward^{t}))$
$\vee(L_{21}^{t} \wedge (\neg FacingWest^{t} \wedge Forward^{t}))$.

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

Truth table method is sound and complete for propositional logic.

Forward, backward chaining are linear-time, complete for Horn clauses

Resolution is complete for propositional logic.

Propositional logic lacks expressive power