Fundamentals of Artificial Intelligence

Logical Agents and Propositional Logic

Knowledge Based Agents

- The ***intelligence of humans*** is achieved not by purely reflex mechanisms but by processes of **reasoning** that operate on internal representations of knowledge.
- In AI, this approach to intelligence is embodied in **knowledge-based agents**.
- **Logical agents** (**knowledge-based agents**) 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*** is supported by logic such as **propositional logic** or **first-order predicate logic**.

Knowledge Base

- A **knowledge base (KB)** is ***a set of sentences in a formal language***.
- **Inference**: Deriving new sentences from old sentences in KB.

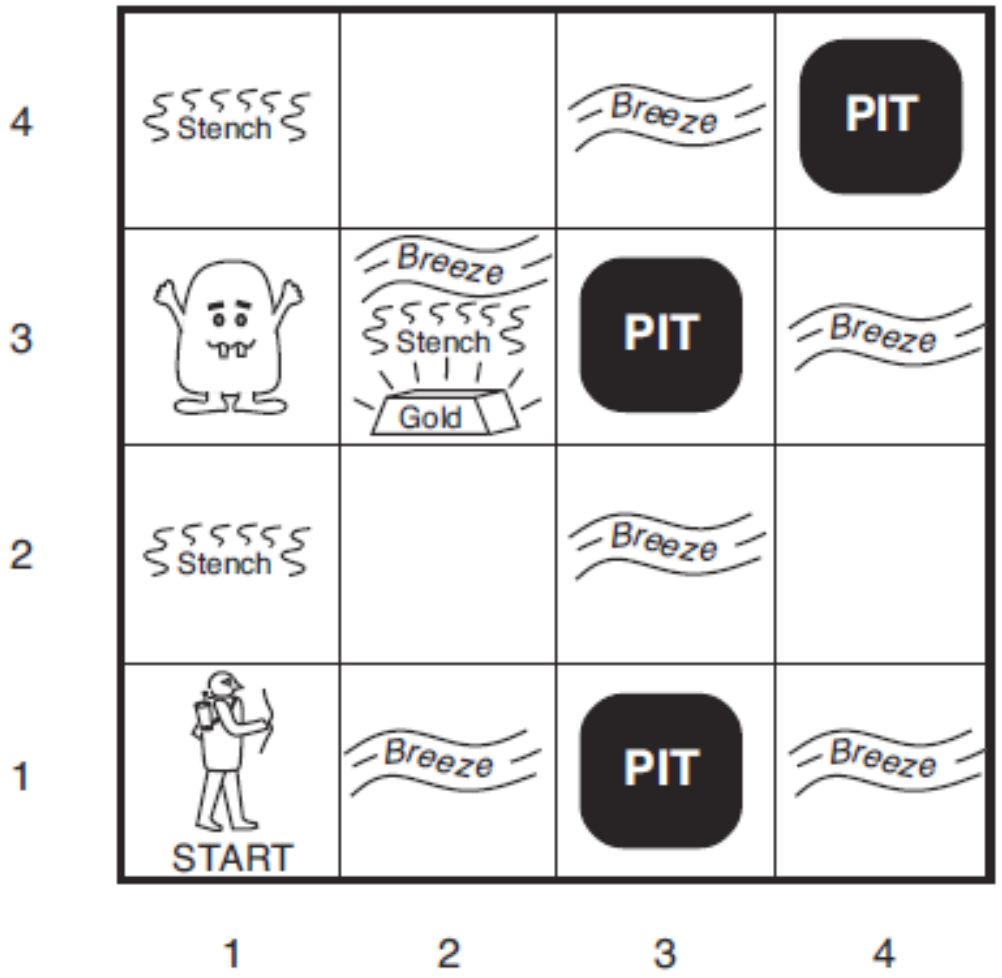
- Each time ***a knowledge-based agent*** does three things.
  1. TELLs the knowledge base what it perceives. TELL operations add new sentences (agent's perceptions) to the knowledge base.
  2. ASKs the knowledge base what action it should perform.
     - Extensive reasoning (inference) may be done about the current state of the world, about the outcomes of possible action sequences.
  3. The agent TELLs the knowledge base which action was chosen, and the agent executes the action.

A Simple Knowledge-Based Agent

```
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
```

- MAKE-PERCEPT-SENTENCE constructs a sentence asserting that the agent perceived the given percept at the given time.
- MAKE-ACTION-QUERY constructs a sentence that asks what action should be done at the current time.
- MAKE-ACTION-SENTENCE constructs a sentence asserting that the chosen action was executed.
- The details of the inference mechanisms are hidden inside TELL and ASK.

Wumpus World



- The **wumpus world** is a simple computer game, but it illustrates some important points about intelligence.
- ***Wumpus*** eats anyone who enters its room.
- The wumpus can be shot by an agent (wumpus dies if it is shot), but the agent has only one arrow.
  - The agent can safely enter a death wumpus's room.
- Some rooms contain bottomless ***pits*** that will trap anyone who wanders into these rooms.
- A room contains **gold**.

Wumpus World Description

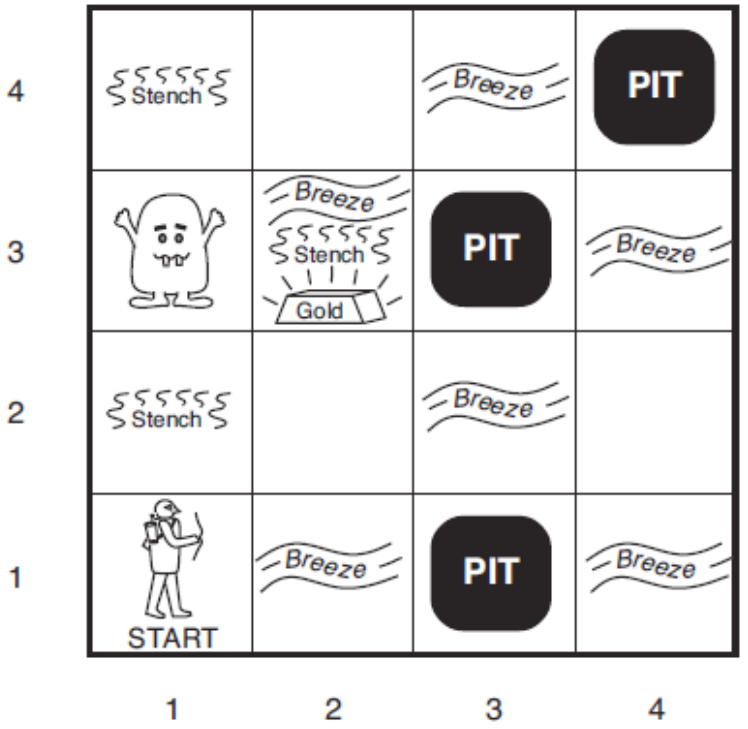
Performance measure: gold +1000, death -1000, -1 per step, -10 for using the arrow

Environment: A 4×4 grid of rooms. The agent always starts in the square [1,1], facing to the right. The locations of the gold and the wumpus are chosen randomly, with a uniform distribution, from the squares other than the start square. Each square other than the start can be a pit, with probability 0.2.

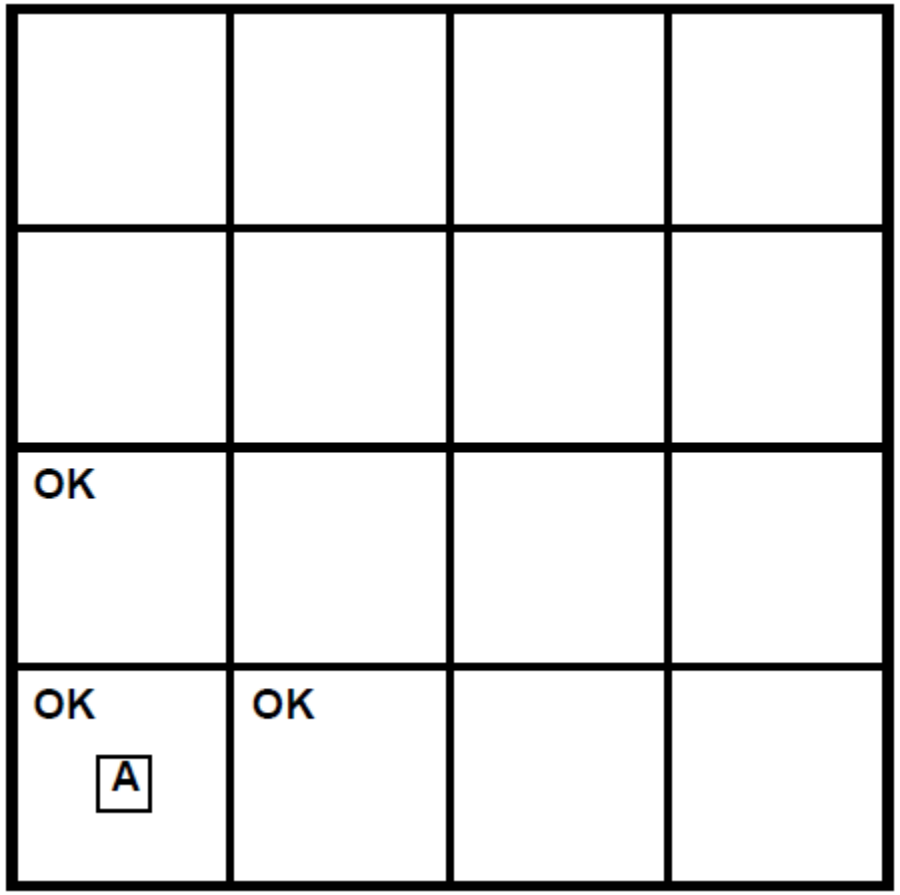
- Squares adjacent to wumpus are smelly
- Squares adjacent to pit are breezy
- Glitter iff gold is in the same square
- Shooting kills wumpus if you are facing it
- Shooting uses up the only arrow
- Grabbing picks up gold if in same square
- Releasing drops the gold in same square

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

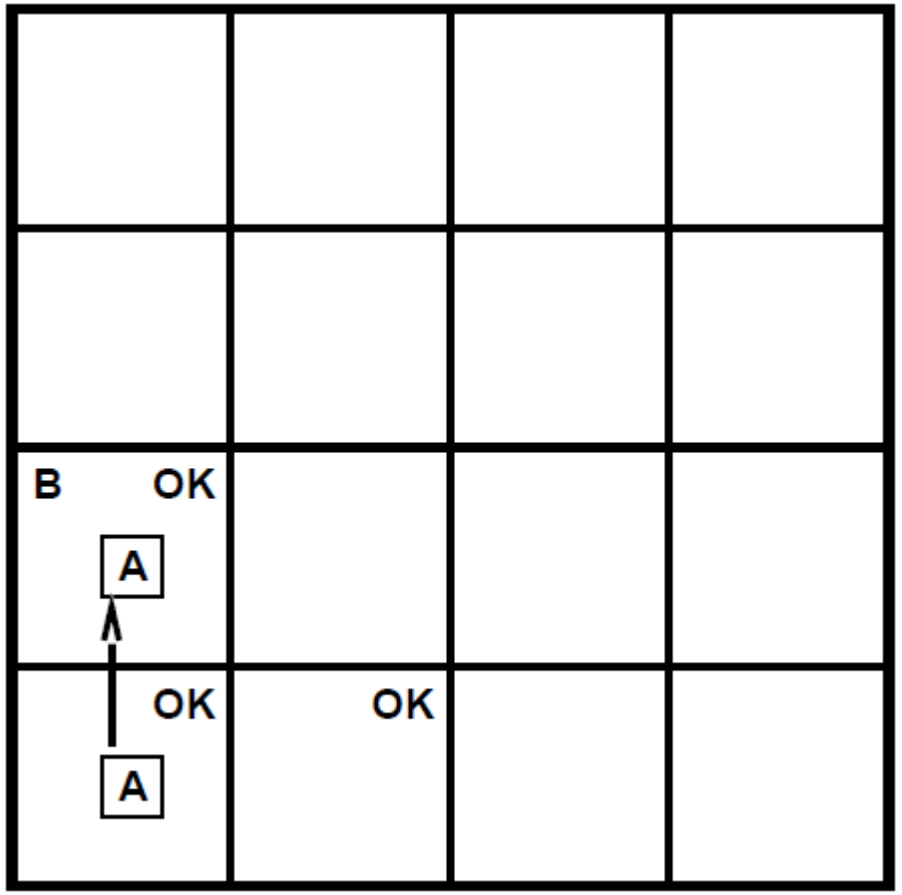
Sensors: Breeze, Glitter, Smell, (Bump, Scream)



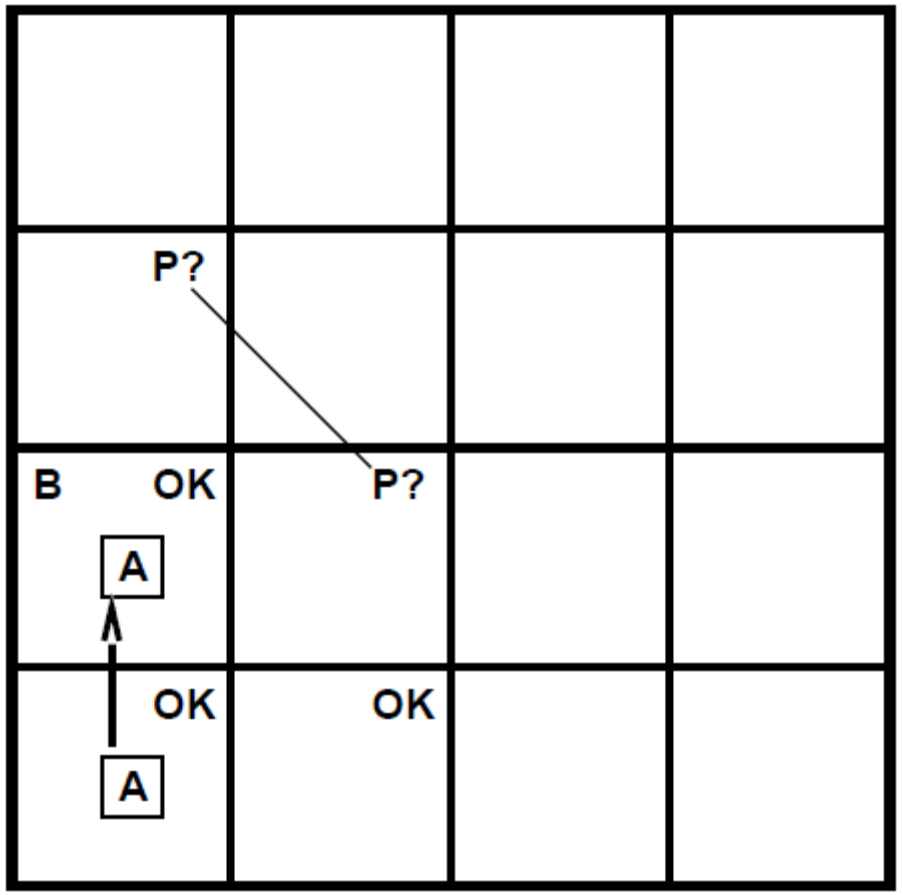
Exploring A Wumpus World



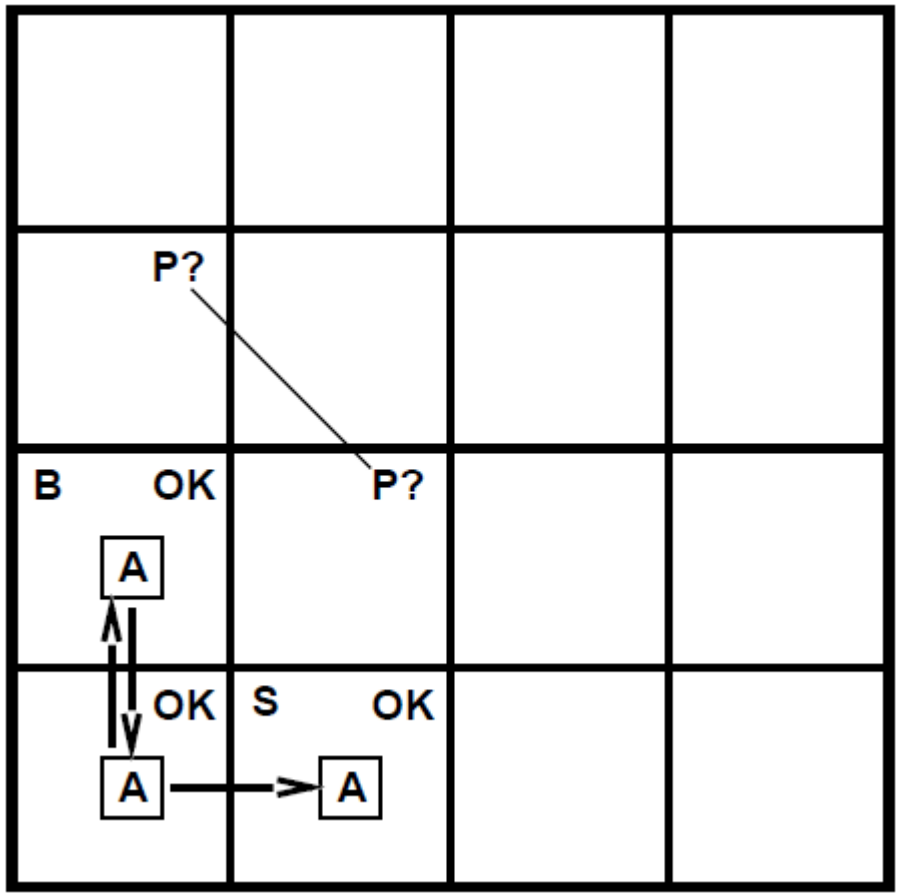
Exploring A Wumpus World



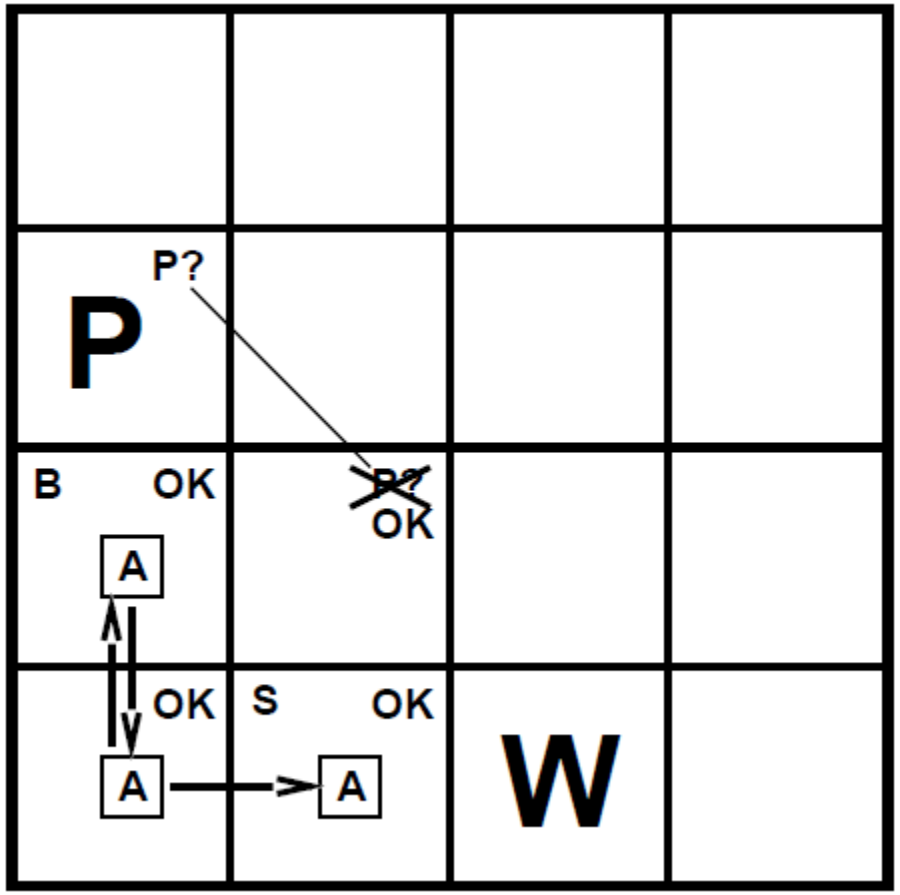
Exploring A Wumpus World



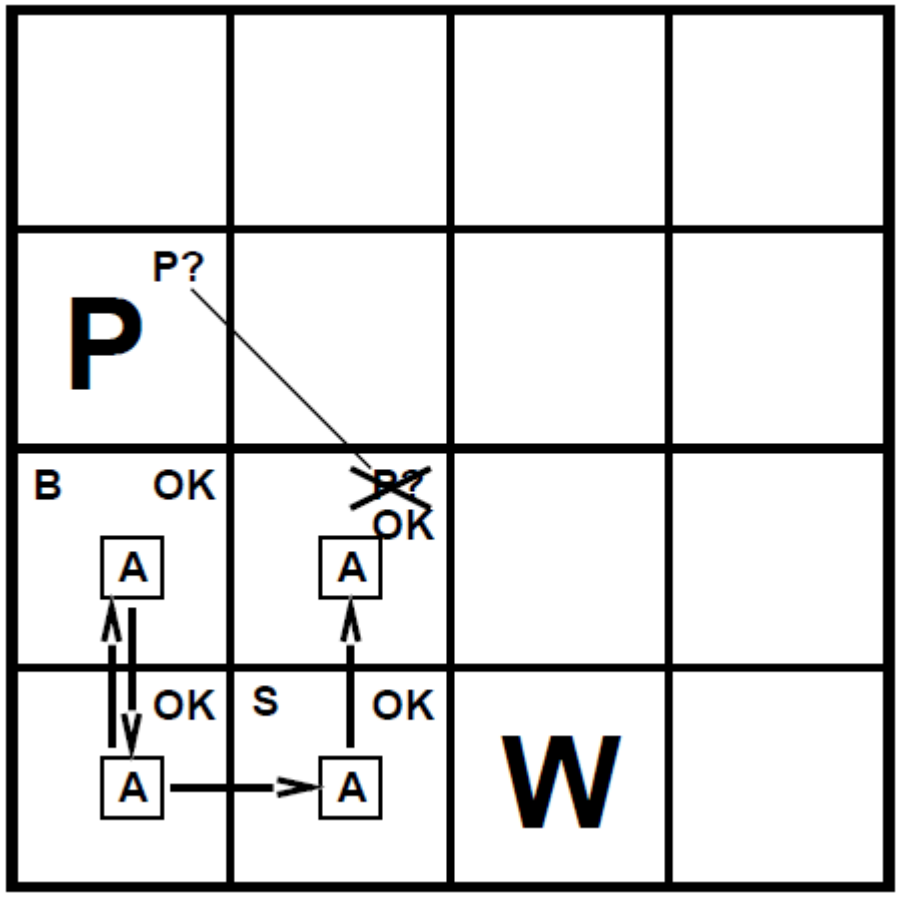
Exploring A Wumpus World



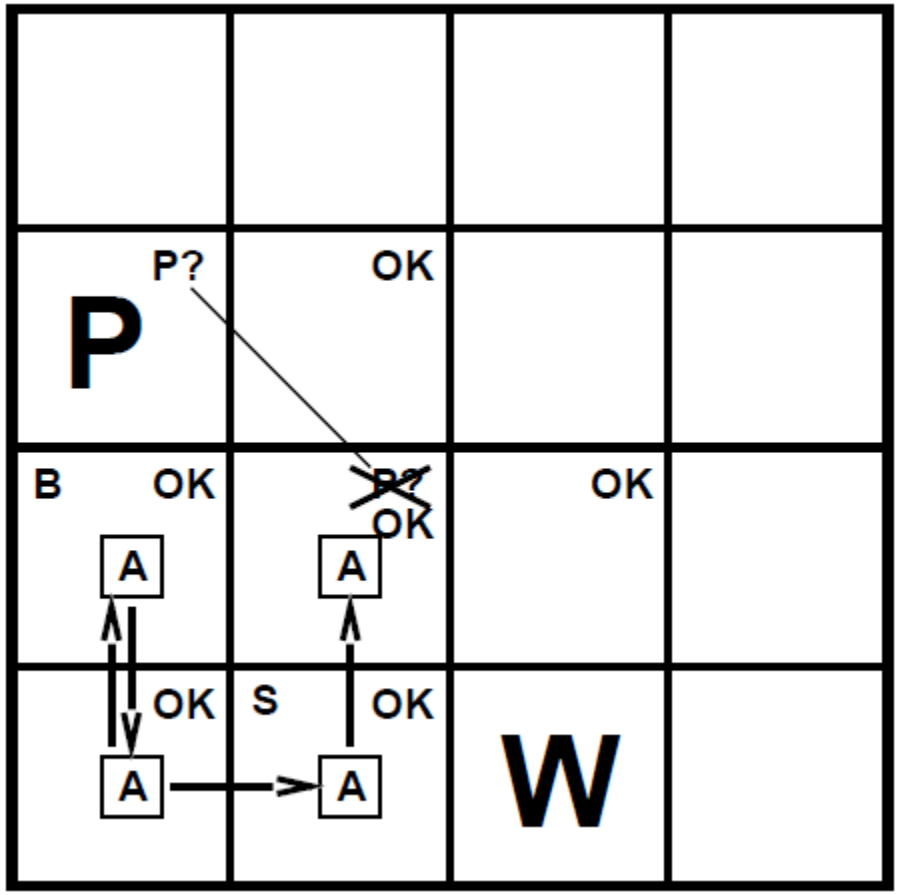
Exploring A Wumpus World



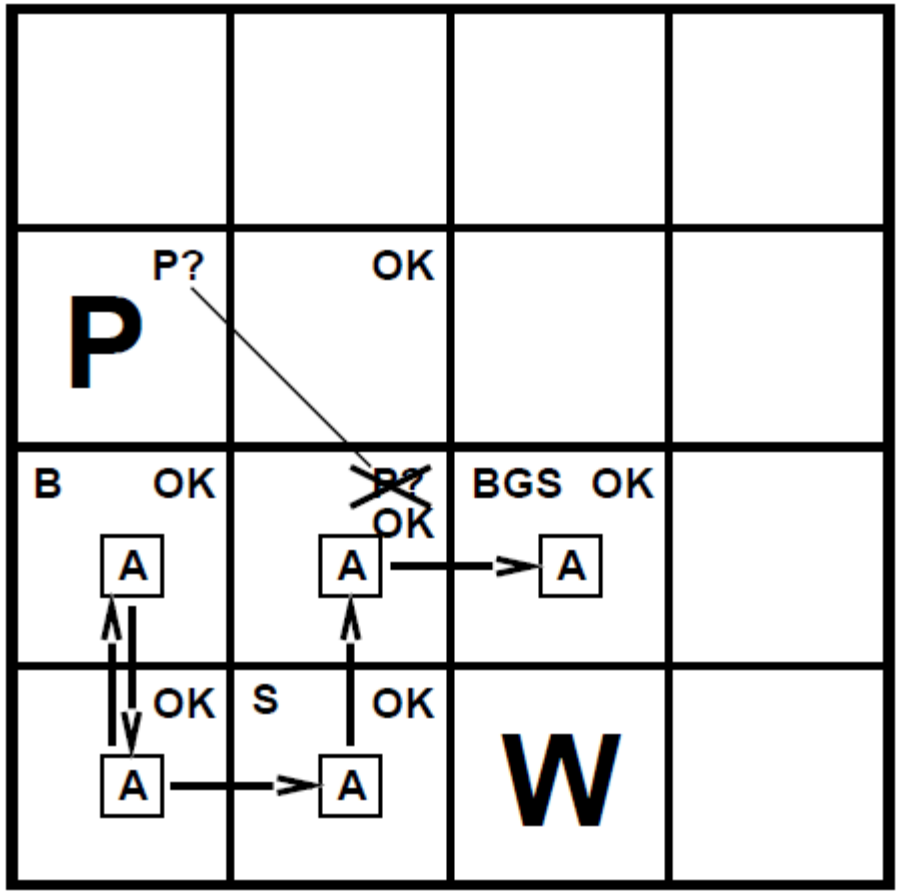
Exploring A Wumpus World



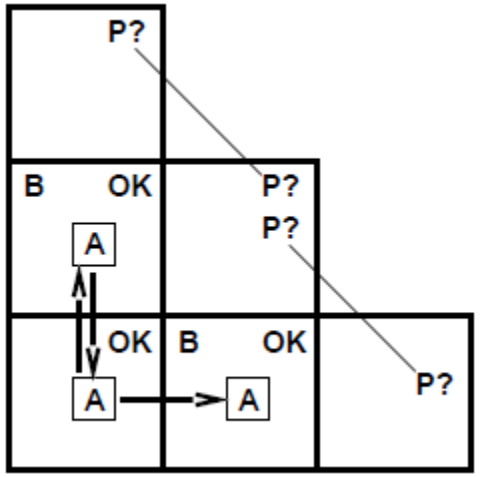
Exploring A Wumpus World



Exploring A Wumpus World

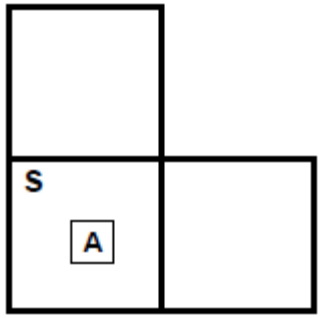


Some Tight Spots in Wumpus World



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

⇒ no safe actions



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

Logic

LOGICS are formal languages for representing information such that conclusions can be drawn.

SYNTAX: The **Syntax** of a logic defines the ***sentences*** (*well-formed formulas*) in that logic.

SEMANTICS: The **Semantics** of a logic defines the **meaning** of sentences.

- The semantics defines the **truth of each sentence** with respect to each **possible world**.
- The term **model** is also used in place of *possible world*.

SATISFACTION:

- **If a sentence α is TRUE in model m, we say that m SATISFIES α (or sometimes we simply say that m is a model of α).**
- The notation M(α) to mean the set of all models of α
  (i.e. **M(α) is the set of all models that satisfy α**) .

Logical Entailment

(other names: Logical Consequence or Logical Implication)

- **Logical Entailment** between sentences means that a sentence follows logically from another sentence. In mathematical notation, $\alpha \models \beta$

The formal definition of entailment :

α logically entails β $\alpha \models \beta$

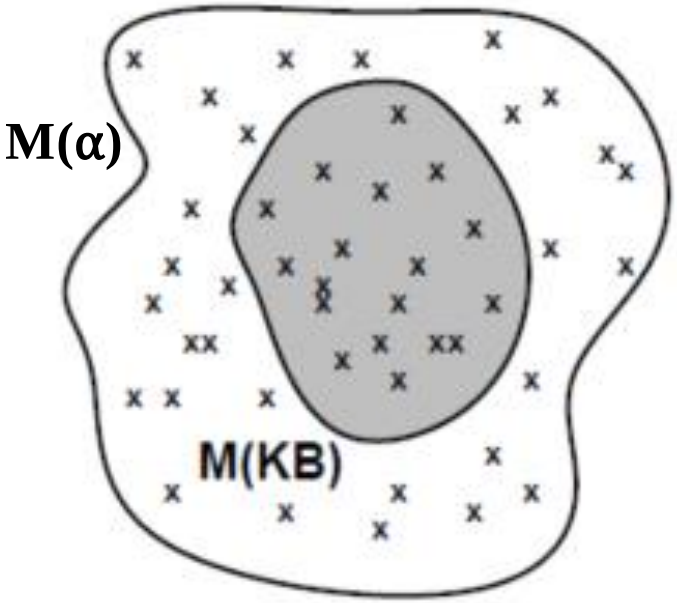
if and only if,

β is true in all worlds where α is true

- $\alpha \models \beta$ if and only if $M(\alpha) \subseteq M(\beta)$.
  - if $\alpha \models \beta$, then α is a *stronger* assertion than β : it rules out *more* possible worlds.

Logical Entailment: 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**
- **M(α) is the set of all models of α (all models satisfy α)**
- Then **KB ⊨ α** if and only if **M(KB) ⊆ M(α)**



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

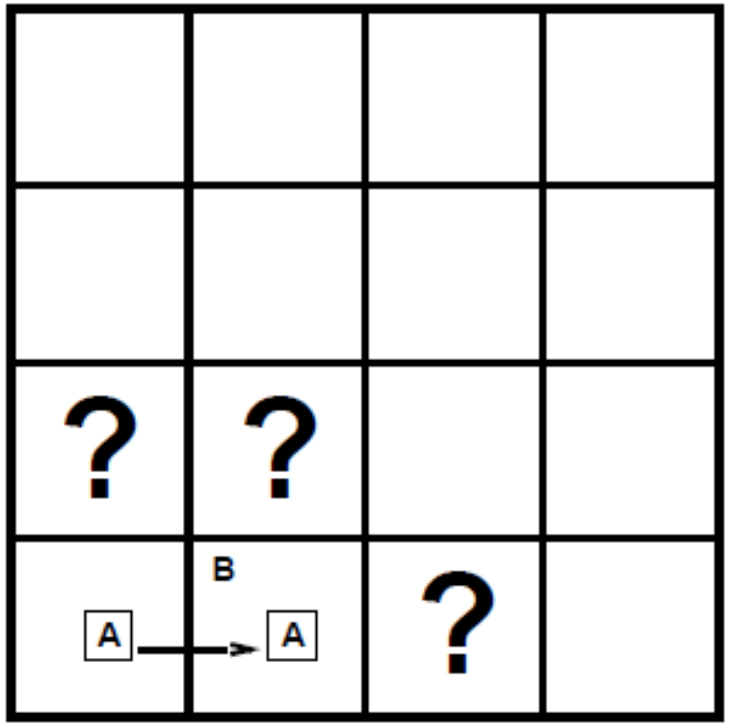
  α = Giants won

Entailment in Wumpus World

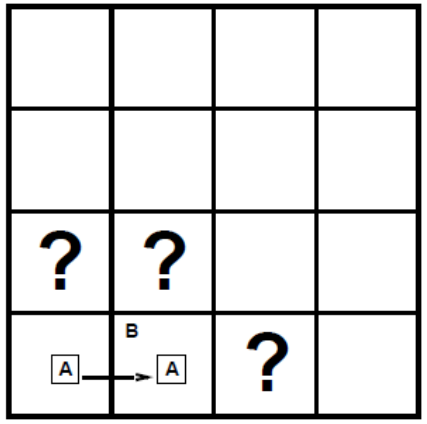
Situation after detecting nothing in [1,1], moving right, breeze in [2,1]

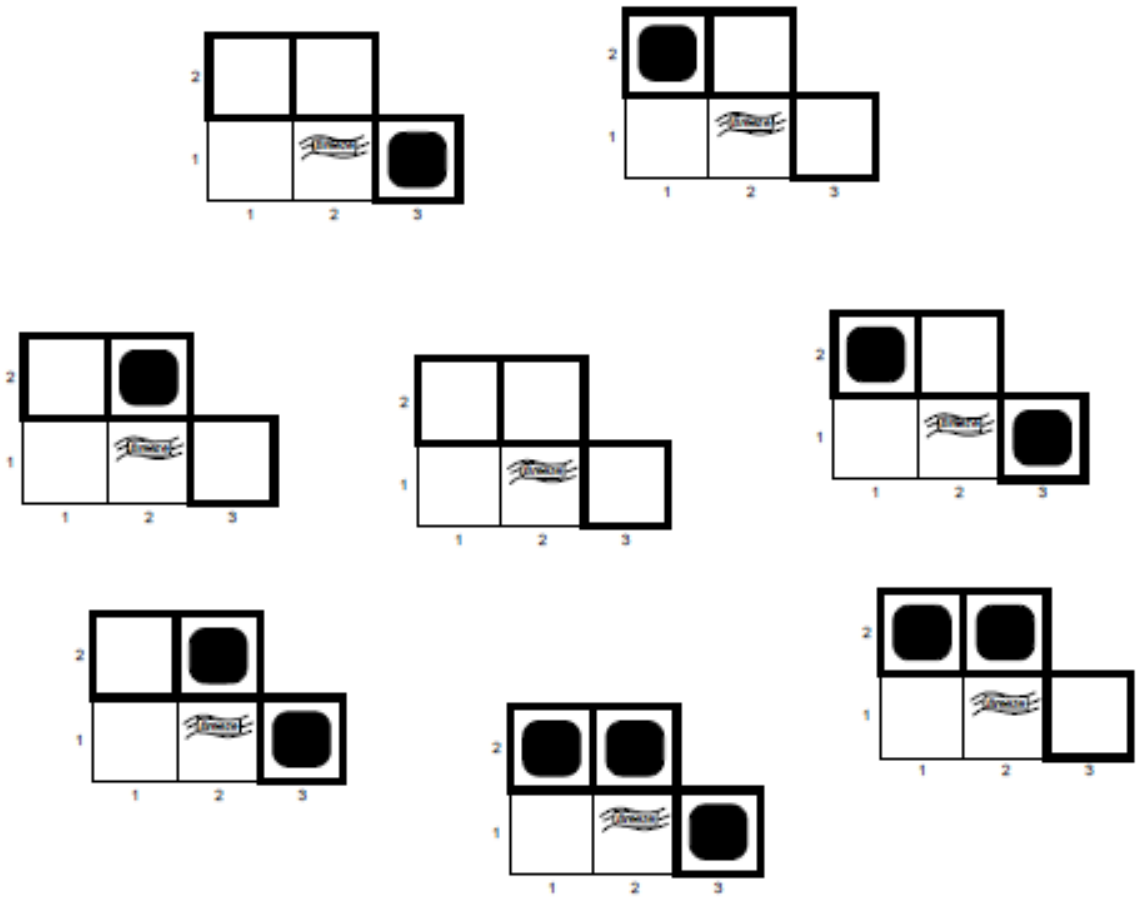
Consider possible models for ?s assuming only pits

3 Boolean choices $\Rightarrow$ 8 possible models



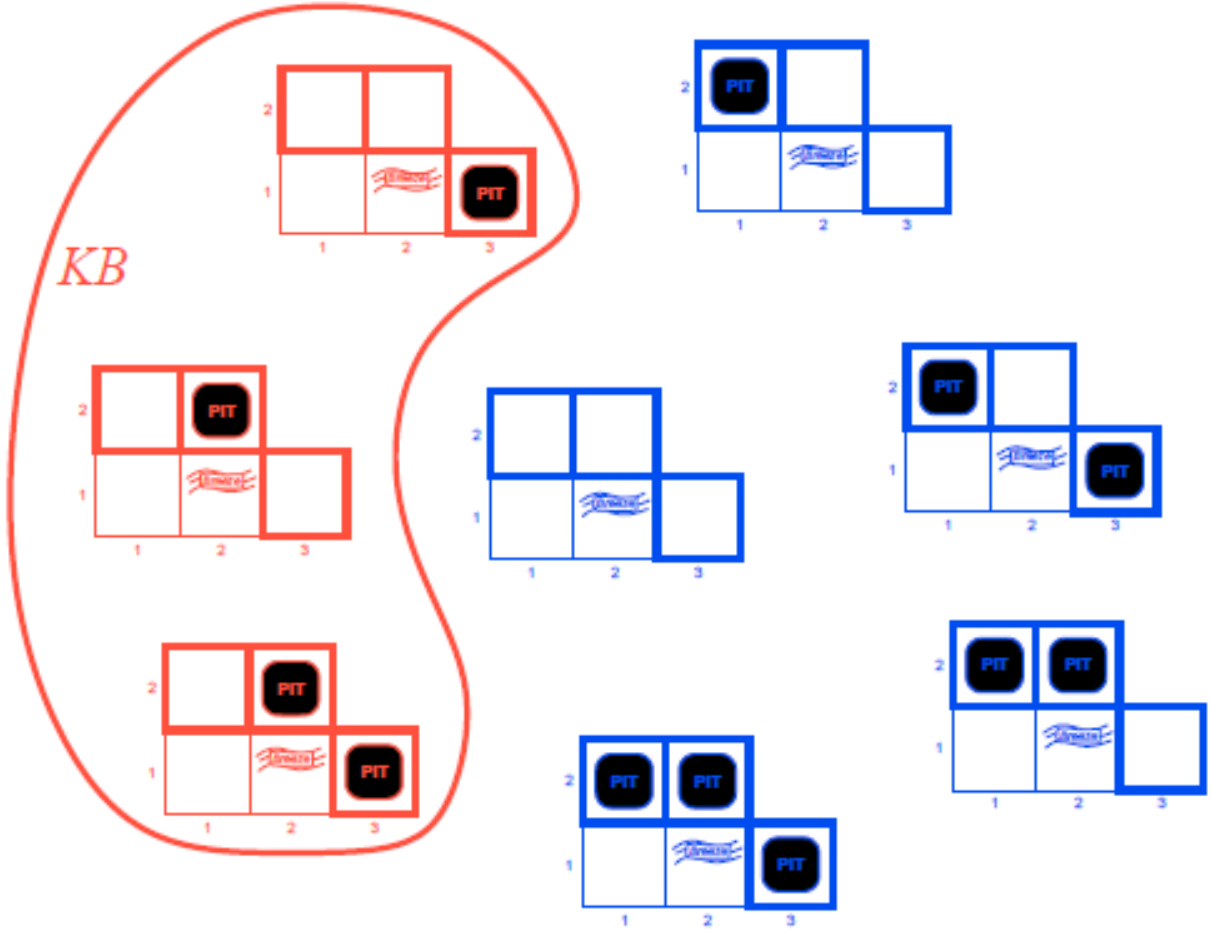
Entailment in Wumpus World: Wumpus Models





Wumpus Models: is [1,2] safe?

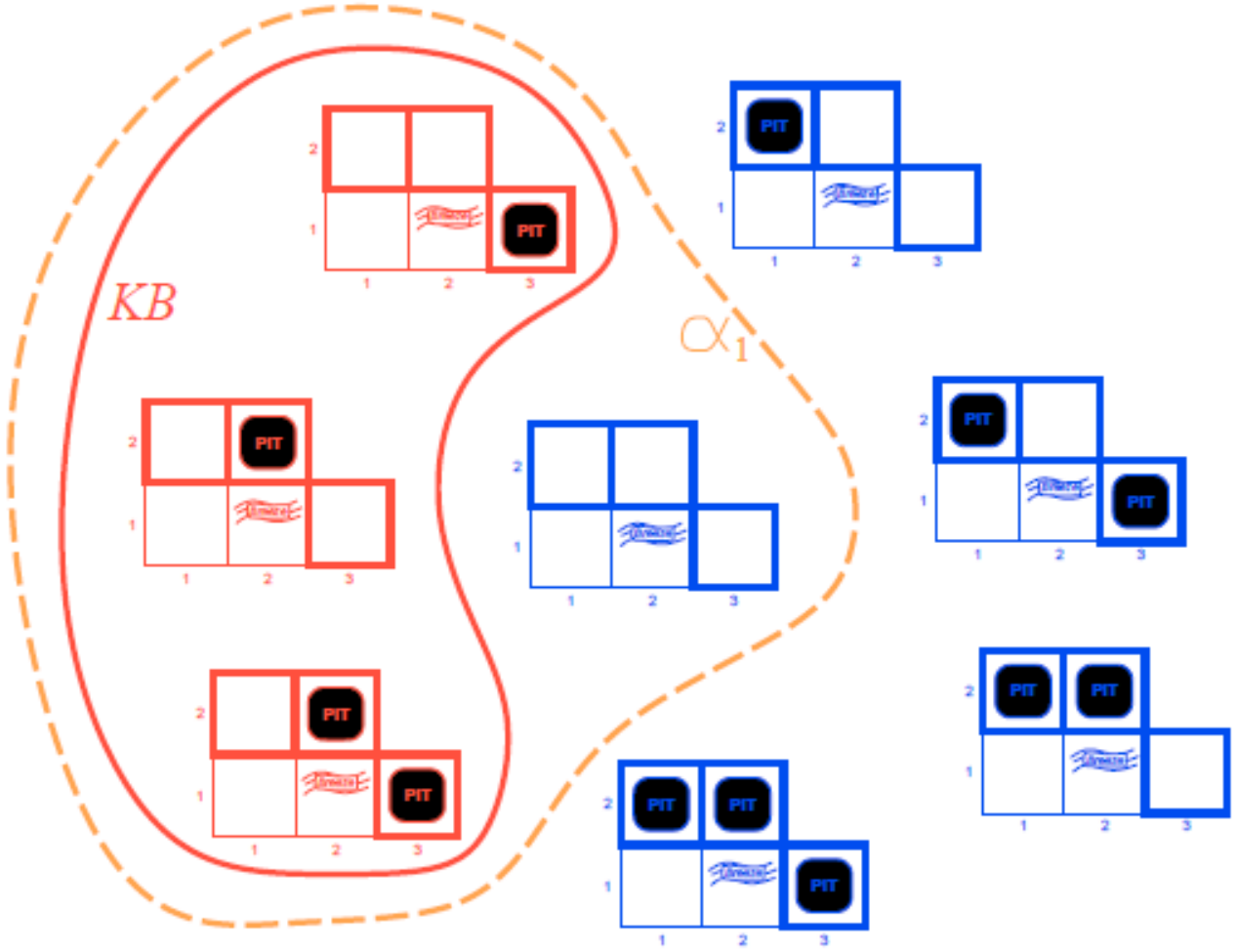
KB = wumpus-world rules + observations



Wumpus Models: is [1,2] safe?

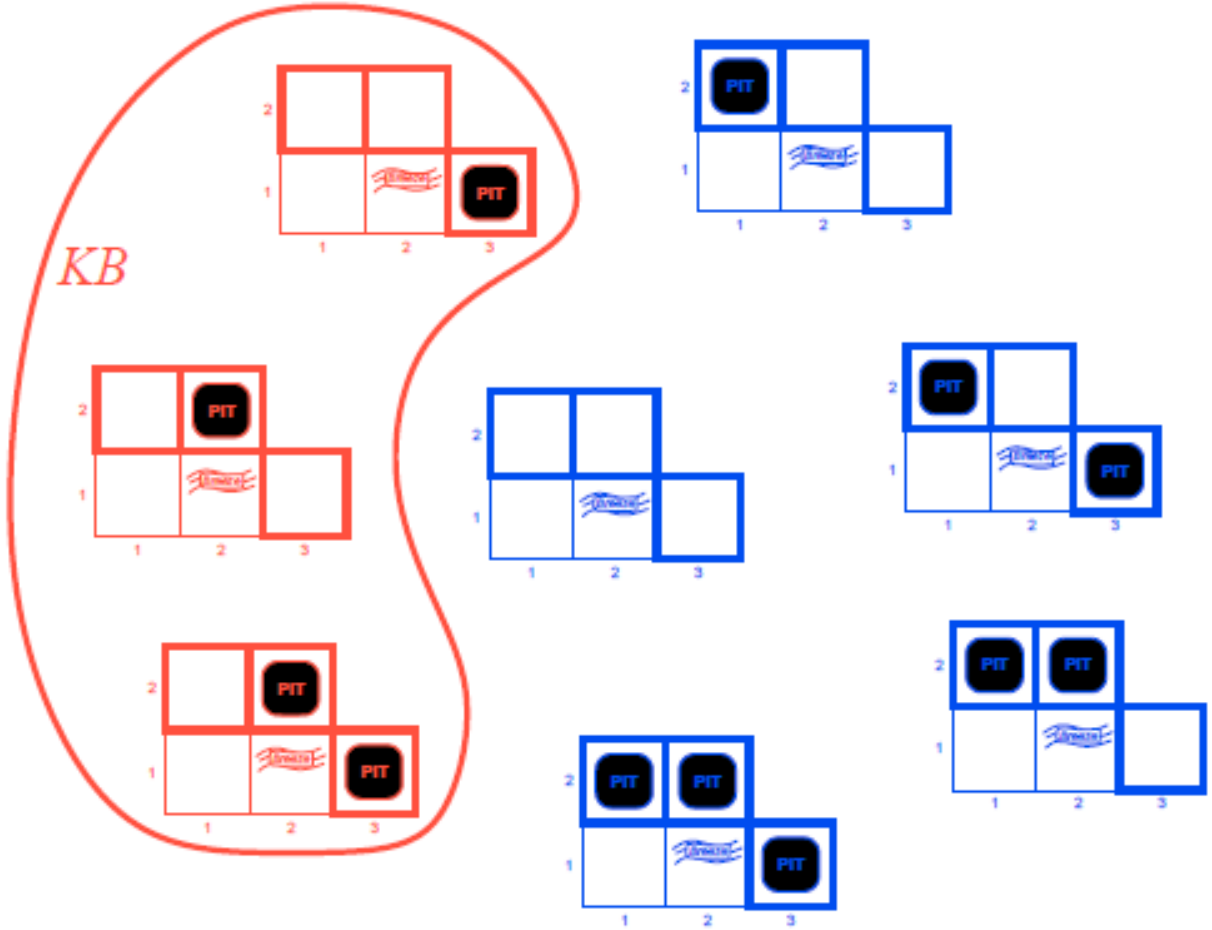
KB = wumpus-world rules + observations

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



Wumpus Models: is [2,2] safe?

KB = wumpus-world rules + observations

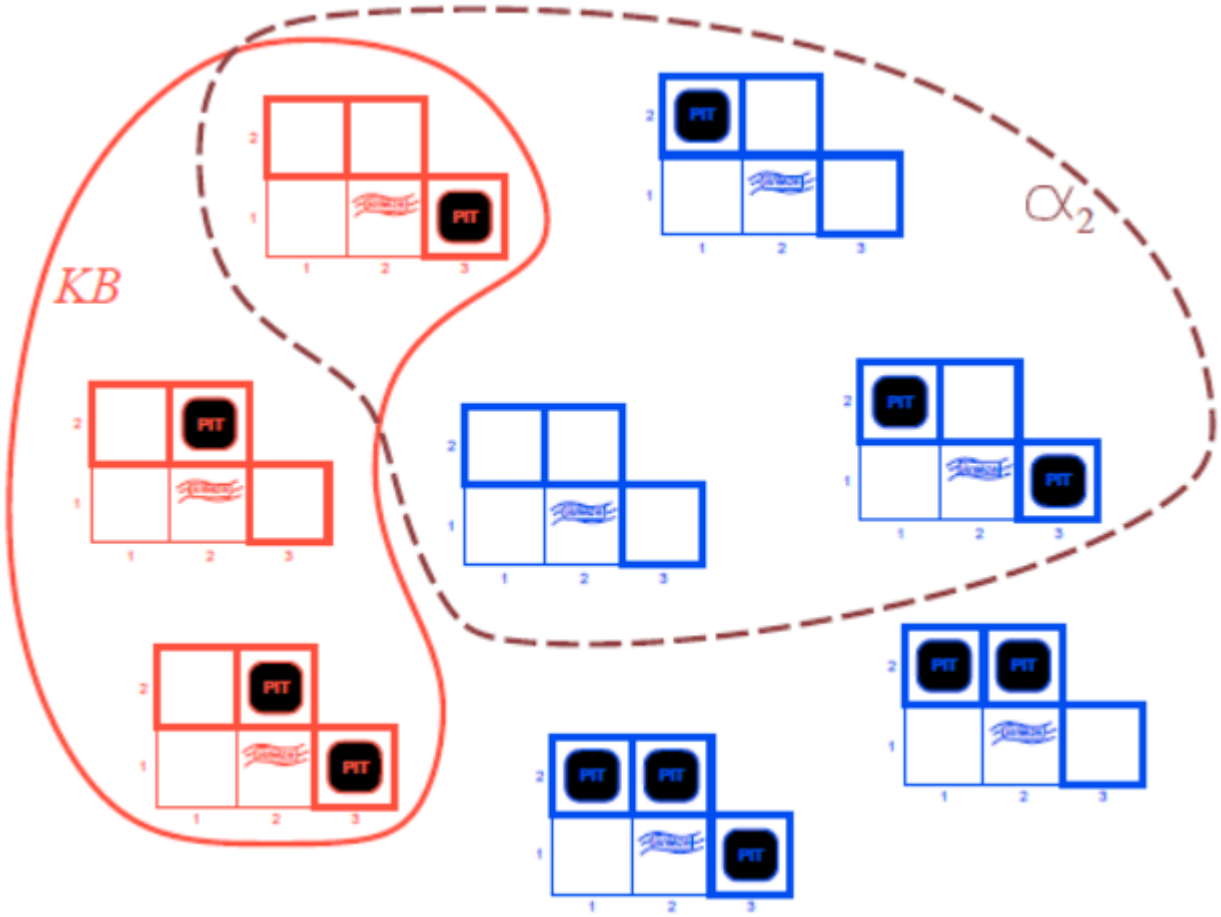


Wumpus Models: is [2,2] safe?

KB = wumpus-world rules + observations

α_2 = "[2,2] is **NOT** safe", KB $\not\models \alpha_2$,
proved by model checking

- the agent cannot conclude that there is no pit in [2,2]. (Nor can it conclude that there is a pit in [2,2].



Model Checking

- In Wumpus world, the ***entailment*** is applied to derive conclusions from KB; i.e. a **logical inference** is carried out by checking the models.
- The algorithm used in this process is called as **model checking**.
- **Model Checking** is an inference algorithm that enumerates all possible models (of KB) to check that α is true in all models in which KB is true, that is, that M(KB) ⊆ M(α). Thus KB ⊨ **α**
- **Model checking** algorithm can be expensive (or impractical) because KB can have too many models (or it can have infinite models in some logics).

Inference (Proofs)

- If an **inference algorithm** i can derive α from KB (generally by syntactic operations), we write

  $$\mathbf{KB} \vdash_i \boldsymbol{\alpha}$$

  which is pronounced "**α is derived from KB** by inference procedure i".

  (or **α is proved/deducted from KB**)

- In a **derivation**, we start from **premises** (sentences (in KB) assumed to be true) and **axioms** (sentences that are true in every world).
- Then, we apply an **inference rule** to sentences that are already derived (proved) to derive a new sentence at each step of the derivation.

Inference: Soundness and Completeness

Soundness:

An inference algorithm i is sound (or truth-preserving) if

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

- If an ***inference algorithm is sound***, all derivable sentences from KB are also **logical consequences** of KB.

Completeness:

An inference algorithm i is complete if

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

- If an ***inference algorithm is complete***, all entailed sentences from KB are also **derivable from** KB.

Propositional Logic

- **Propositional Logic** is a simple but powerful logic.
- The **syntax of propositional logic** defines the ***allowable sentences***.
- The **semantics of propositional logic** defines the rules for ***determining the truth of a sentence*** with respect to a particular model.
- Propositional Logic contains ***proposition symbols*** and ***logical connectives (logical operators).***
  - The ***meaning of a propositional symbol*** can be **true** or **false**.
  - The ***meaning of a logical connective*** is given by its truth table.

Propositional Logic: Syntax

$$
\begin{aligned}
\mathit{Sentence} &\rightarrow \mathit{AtomicSentence} \mid \mathit{ComplexSentence} \\
\mathit{AtomicSentence} &\rightarrow \mathit{True} \mid \mathit{False} \mid P \mid Q \mid R \mid \ldots \\
\mathit{ComplexSentence} &\rightarrow (\ \mathit{Sentence}\ ) \mid [\ \mathit{Sentence}\ ] \\
&\mid \neg\, \mathit{Sentence} \qquad \text{NEGATION} \\
&\mid \mathit{Sentence} \wedge \mathit{Sentence} \qquad \text{CONJUNCTION} \\
&\mid \mathit{Sentence} \vee \mathit{Sentence} \qquad \text{DISJUNCTION} \\
&\mid \mathit{Sentence} \Rightarrow \mathit{Sentence} \qquad \text{IMPLICATION} \\
&\mid \mathit{Sentence} \Leftrightarrow \mathit{Sentence} \qquad \text{BICONDITIONAL}
\end{aligned}
$$

- The **atomic sentences** consist of a single **proposition symbol**.
  - Each ***proposition symbol*** stands for a proposition that can be **true** or **false**.
  - There are two proposition symbols with fixed meanings (constants): **True** is the always-true proposition and **False** is the always-false proposition.
- **Complex sentences** are constructed from other sentences, using parentheses and logical connectives.

Propositional Logic: Syntax

- A **literal** is either an ***atomic sentence*** (a **positive literal**) or ***a negated atomic sentence*** (a **negative literal**).
- A sentence whose main connective is $\wedge$, such as $P \wedge Q$, is called a **conjunction**; its parts are the **conjuncts**.
- A sentence whose main connective is $\vee$, such as $P \vee Q$, is called a **disjunction**; its parts are the **disjuncts**.
- A sentence such as $P \Rightarrow Q$ is called an **implication** (or conditional). Its **premise** or **antecedent** is P, and its **conclusion** or **consequent** is Q.

OPERATOR PRECEDENCE : $\neg, \wedge, \vee, \Rightarrow, \Leftrightarrow$

Propositional Logic: Semantics

- The **semantics** defines the rules for determining the ***truth of a sentence*** with respect to a particular ***model***.
- In propositional logic, a **model** simply fixes the ***truth value***—true or false—for every proposition symbol.

Ex. Model:

- If the sentences in the knowledge base make use of the proposition symbols P, Q, and R, then one possible model is

  m_1 = {P=false, Q=false, R=true} .
- With three proposition symbols, there are 2^3=8 possible models

P	Q	R
true	true	true
true	true	false
true	false	true
true	false	false
false	true	true
false	true	false
false	false	true
false	false	false

Propositional Logic: Semantics

- The **semantics for propositional logic** must specify ***how to compute the truth value of any sentence***, for a given a model m.

- Since all sentences are constructed from ***atomic sentences*** and the ***five connectives***; we need to specify
  - how to compute the truth of atomic sentences and
  - how to compute the truth of sentences formed with each of the five connectives.

Propositional Logic: Semantics

Rules for evaluating truth with respect to a model m

Atomic Sentences:

- **True** is true in every model and **False** is false in every model.
- The ***truth value of every other proposition symbol*** must be specified directly in the model m.

Complex Sentences: five rules which hold for any sub-sentences P and Q in any model m

- ¬P is true iff P is false in m.
- P ∧ Q is true iff both P and Q are true in m.
- P ∨ Q is true iff either P or Q is true in m.
- P ⇒ Q is true unless P is true and Q is false in m.
- P ⇔ Q is true iff P and Q are both true or both false in m.

Propositional Logic: Semantics

Truth Tables

- The ***semantic rules*** can also be expressed with **truth tables** that specify the truth value of a complex sentence for each possible assignment of truth values to its components.

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*

Propositional Logic

Wumpus world sentences

- $P_{x,y}$ is true if there is a pit in [x, y].
- $W_{x,y}$ is true if there is a wumpus in [x, y], dead or alive.
- $B_{x,y}$ is true if the agent perceives a breeze in [x, y].
- $S_{x,y}$ is true if the agent perceives a stench in [x, y].

- There is no pit in [1,1]: $R_1 : \quad \neg P_{1,1}$
- A square is breezy if and only if there is a pit in a neighboring square.

$$R_2 : \quad B_{1,1} \quad \Leftrightarrow \quad (P_{1,2} \vee P_{2,1}) \, .$$
$$R_3 : \quad B_{2,1} \quad \Leftrightarrow \quad (P_{1,1} \vee P_{2,2} \vee P_{3,1})$$

- The breeze percepts for the first two squares visited in the specific world the agent is in.

$$R_4 : \quad \neg B_{1,1}$$
$$R_5 : \quad B_{2,1} \, .$$

- From these sentences, we can derive $\neg P_{1,2}$ (there is no pit in [1,2]).

Propositional Logic
Inference by Enumeration of Models

- Our goal now is to decide whether KB $\models \alpha$ for some sentence α.
- For example, is $\neg P_{1,2}$ entailed by our KB?

- ***Inference by Enumeration of Models*** is a direct implementation of the definition of entailment:
  - Enumerate the models, and check that α is true in every model in which KB is true.
  - Models are assignments of true or false to every proposition symbol.

- Inference by Enumeration of Models algorithm is **sound** and **complete**.
- If KB and α contain n symbols in all, then there are 2^n models.
  - Thus, the time complexity of the algorithm is $O(2^n)$.

Propositional Logic

Inference by Enumeration of Models

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

  $symbols \leftarrow$ a list of the proposition symbols in KB and α
  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 // when KB is false, always return true*
  else do
    $P \leftarrow$ FIRST($symbols$)
    $rest \leftarrow$ REST($symbols$)
    return (TT-CHECK-ALL($KB, \alpha, rest, model \cup \{P = true\}$)
      and
      TT-CHECK-ALL($KB, \alpha, rest, model \cup \{P = false\}$))

Propositional Logic

Inference by Enumeration of Models

$B_{1,1}$	$B_{2,1}$	$P_{1,1}$	$P_{1,2}$	$P_{2,1}$	$P_{2,2}$	$P_{3,1}$	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
- Since there are 7 propositional symbols, there are 128 rows.

- In all 3 rows, $P_{1,2}$ is false, so there is no pit in [1,2].
- On the other hand, there might (or might not) be a pit in [2,2].

Logical Equivalence

Two sentences are logically equivalent iff true in same models:

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

$$
\begin{aligned}
(\alpha \wedge \beta) &\equiv (\beta \wedge \alpha) && \text{commutativity of } \wedge \\
(\alpha \vee \beta) &\equiv (\beta \vee \alpha) && \text{commutativity of } \vee \\
((\alpha \wedge \beta) \wedge \gamma) &\equiv (\alpha \wedge (\beta \wedge \gamma)) && \text{associativity of } \wedge \\
((\alpha \vee \beta) \vee \gamma) &\equiv (\alpha \vee (\beta \vee \gamma)) && \text{associativity of } \vee \\
\neg(\neg\alpha) &\equiv \alpha && \text{double-negation elimination} \\
(\alpha \Rightarrow \beta) &\equiv (\neg\beta \Rightarrow \neg\alpha) && \text{contraposition} \\
(\alpha \Rightarrow \beta) &\equiv (\neg\alpha \vee \beta) && \text{implication elimination} \\
(\alpha \Leftrightarrow \beta) &\equiv ((\alpha \Rightarrow \beta) \wedge (\beta \Rightarrow \alpha)) && \text{biconditional elimination} \\
\neg(\alpha \wedge \beta) &\equiv (\neg\alpha \vee \neg\beta) && \text{De Morgan} \\
\neg(\alpha \vee \beta) &\equiv (\neg\alpha \wedge \neg\beta) && \text{De Morgan} \\
(\alpha \wedge (\beta \vee \gamma)) &\equiv ((\alpha \wedge \beta) \vee (\alpha \wedge \gamma)) && \text{distributivity of } \wedge \text{ over } \vee \\
(\alpha \vee (\beta \wedge \gamma)) &\equiv ((\alpha \vee \beta) \wedge (\alpha \vee \gamma)) && \text{distributivity of } \vee \text{ over } \wedge
\end{aligned}
$$

Validity and Satisfiability

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

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

Validity is connected to inference via the Deduction Theorem:

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

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 α by $reductio\ ad\ absurdum$

Inference and Proofs

- A **proof** is a sequence of sentences that leads to the desired goal.
- A proof starts with given ***premises*** and each new sentence in the sequence is obtained as a result of the application of an **inference rule** to previous sentences in the sequence.

Some Inference Rules:

Modus Ponens:

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

And-Elimination:

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

- All logical equivalences can be used as inference rules.

$$\frac{\alpha \Leftrightarrow \beta}{(\alpha \Rightarrow \beta) \wedge (\beta \Rightarrow \alpha)} \quad \text{and} \quad \frac{(\alpha \Rightarrow \beta) \wedge (\beta \Rightarrow \alpha)}{\alpha \Leftrightarrow \beta}$$

Inference and Proofs

- ***Inference rules*** are **sound**. i.e. when their *premises* are ***true*** their *conclusions* are also ***true.***

Wumpus World Proof Example:

1.	(WumpusAhead ∧WumpusAlive) ⇒ Shoot	Premise
2.	(WumpusAhead ∧ WumpusAlive)	Premise
3.	Shoot	Modus Ponens 1 and 2

Another Example:

1.	(WumpusAhead ∧ WumpusAlive)	Premise
2.	WumpusAlive	And-Elimination 1

Inference and Proofs

Derivation (proof) of $\neg P_{1,2}$ from given premises

1.	$\neg P_{1,1}$	Premise
2.	$B_{1,1} \Leftrightarrow (P_{1,2} \vee P_{2,1})$	Premise
3.	$B_{2,1} \Leftrightarrow (P_{1,1} \vee P_{2,2} \vee P_{3,1})$	Premise
4.	$\neg B_{1,1}$	Premise
5.	$B_{2,1}$	Premise
6.	$(B_{1,1} \Rightarrow (P_{1,2} \vee P_{2,1})) \wedge ((P_{1,2} \vee P_{2,1}) \Rightarrow B_{1,1})$	Biconditional elimination to 2
7.	$((P_{1,2} \vee P_{2,1}) \Rightarrow B_{1,1})$	And-Elimination to 6
8.	$(\neg B_{1,1} \Rightarrow \neg(P_{1,2} \vee P_{2,1}))$	Logical equivalence for contrapositives to 7
9.	$\neg(P_{1,2} \vee P_{2,1})$	Modus Ponens to 8 and 4
10.	$\neg P_{1,2} \wedge \neg P_{2,1}$	De Morgan’s rule to 9
11.	$\neg P_{1,2}$	And-Elimination to 10

Searching for Proofs

- Searching for proofs is an alternative to enumerating models (exponential) and it is more efficient.
- We can apply any of the search algorithms to find a sequence of steps that constitutes a proof. We just need to define a proof problem as follows:

INITIAL STATE: The initial knowledge base.

ACTIONS: The set of actions consists of all the inference rules applied to all the sentences that match the top half of the inference rule.

RESULT: The result of an action is to add the sentence in the bottom half of the inference rule.

GOAL: The goal is a state that contains the sentence we are trying to prove.

Proof by Resolution

- The inference rules that we look at are **sound**, but we have not discussed the question of **completeness** for the inference algorithms that use them.
- Search algorithms such as *iterative deepening search* are **complete** in the sense that they will find any reachable goal.
  - But if the available inference rules are ***inadequate***, then the goal is not reachable—no proof exists that uses only those inference rules.
  - For example, if we removed the biconditional elimination rule, the Wumpus World proof would not go through.
  - So, our proof system will **not** be **complete**.

- A ***single inference rule***, **RESOLUTION**, that yields a complete inference algorithm when coupled with any complete search algorithm.

Clause

- **Resolution** inference rule works on **clauses**.
- A **clause** is a ***disjunction of literals.***
  - A **literal** is either an ***atomic sentence*** (a **positive literal**) or ***a negated atomic sentence*** (a **negative literal)**
- Clause Example:

  $$P \vee R \vee \neg Q \vee \neg S$$

- A single literal can be viewed as a disjunction of one literal, also known as a **unit clause**.

- **Resolution** is **sound** and **complete** for propositional logic

Resolution

Resolution Inference Rule

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

- where l_i and m_j are ***complementary literals***. The resolution takes two clauses and produces a new clause containing all the literals of the two original clauses *except* the two complementary literals.
- Conclusion of the rule is called as **resolvent**.

P ∨ R ∨ ¬Q ∨ ¬S T ∨ Q ∨ ¬M

P ∨ R ∨ ¬S ∨ T ∨ ¬M

Conjunctive Normal Form

- The resolution rule applies only to clauses.
  - knowledge bases and queries should consist of clauses.
- Every sentence of propositional logic is logically equivalent to a **conjunction of clauses**.
- A sentence expressed as a *conjunction of clauses* is said to be in **conjunctive normal form (CNF).**

Converting to CNF

1. Eliminate $\Leftrightarrow$, replacing $\alpha \Leftrightarrow \beta$ with $(\alpha \Rightarrow \beta) \wedge (\beta \Rightarrow \alpha)$

2. Eliminate $\Rightarrow$, replacing $\alpha \Rightarrow \beta$ with $\neg\alpha \vee \beta$

3. CNF requires $\neg$ to appear only in literals, so we "move $\neg$ inwards" by repeated application of the following equivalences

$$\neg(\neg\alpha) \equiv \alpha \quad \text{(double-negation elimination)}$$
$$\neg(\alpha \wedge \beta) \equiv (\neg\alpha \vee \neg\beta) \quad \text{(De Morgan)}$$
$$\neg(\alpha \vee \beta) \equiv (\neg\alpha \wedge \neg\beta) \quad \text{(De Morgan)}$$

4. Now we have a sentence containing nested $\wedge$ and $\vee$ operators applied to literals. We apply the distributivity law, distributing $\vee$ over $\wedge$ wherever possible.

Converting to CNF

- Example:

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

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

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

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

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

A Simple Resolution Algorithm for Propositional Logic

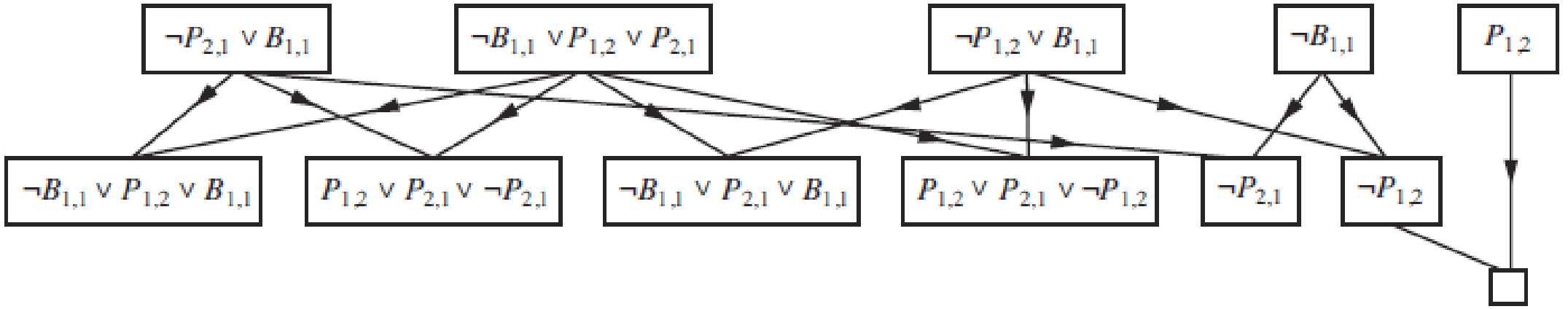
function PL-RESOLUTION(KB, α) **returns** *true* or *false*
  inputs: KB, the knowledge base, a sentence in propositional logic
    α, the query, a sentence in propositional logic

  $clauses \leftarrow$ the set of clauses in the CNF representation of $KB \wedge \neg\alpha$
  $new \leftarrow \{\ \}$
  loop do
    for each pair of clauses 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*
    $clauses \leftarrow clauses \cup new$

- PL-RESOLVE returns the set of all possible clauses obtained by resolving its two inputs.

PL-RESOLUTION Example

- Partial application of PL-RESOLUTION to a simple inference in the Wumpus world. $\neg P_{1,2}$ is shown to follow from the first four clauses in the top row.



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

Horn Clauses and Definite Clauses

- The **definite clause**, which is a disjunction of literals of which ***exactly one is positive***.
- **Horn clause** is a disjunction of literals of which ***at most one is positive***.
  - So all **definite clauses** are **Horn clause**s,
  - Clauses with no positive literals; these are called **goal clauses**.
- Knowledge bases containing only definite clauses are interesting for three reasons:
  1. Every definite clause can be written as an ***implication*** whose premise is a conjunction of positive literals and whose conclusion is a single positive literal.
     - In Horn form, the *premise* is called the **body** and the *conclusion* is called the **head**.
     - A sentence consisting of a single positive literal is called a **fact**.
  2. Inference with Horn clauses can be done through the **forward chaining** and **backward chaining** algorithms. This type of inference is the basis for **logic programming.**
  3. Deciding entailment with Horn clauses can be done in time that is **linear** in the size of the knowledge base

Horn Clauses and Definite Clauses

Definite Clause Examples:

$P \vee \neg R \vee \neg S$	$R \wedge S \Rightarrow P$
P	P

Goal Clause Example:

$\neg R \vee \neg S$	$R \wedge S \Rightarrow \text{False}$ or $R \wedge S \Rightarrow$

Forward-Chaining Algorithm for Propositional Logic

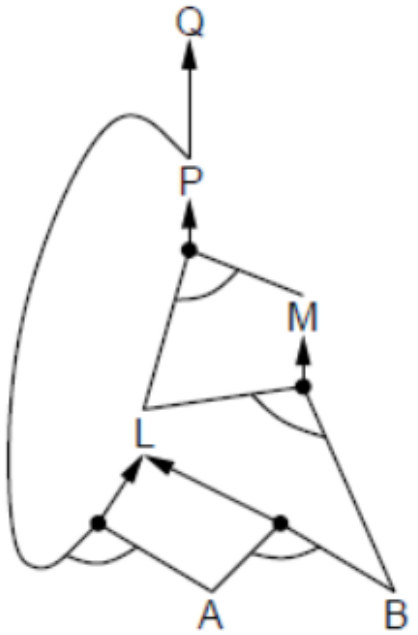
```
function PL-FC-ENTAILS?(KB, q) returns true or false
  inputs: KB, the knowledge base, a set of propositional definite clauses
          q, the query, a proposition symbol
  count ← a table, where count[c] is the number of symbols in c's premise
  inferred ← a table, where inferred[s] is initially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

Forward-Chaining Algorithm Example

FC Algorithm: Fire any rule whose premises are satisfied in the KB, add its conclusion to the KB, until query is found.

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

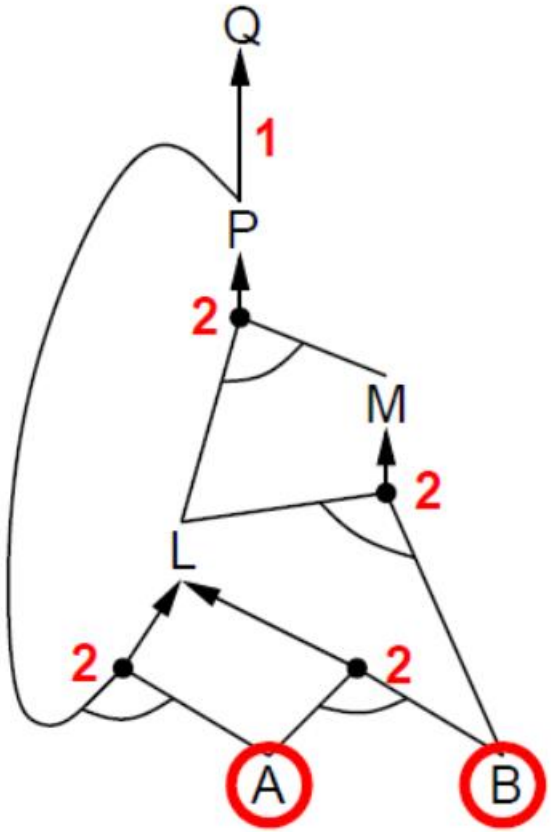
A set of Horn clauses.



The corresponding AND–OR graph.

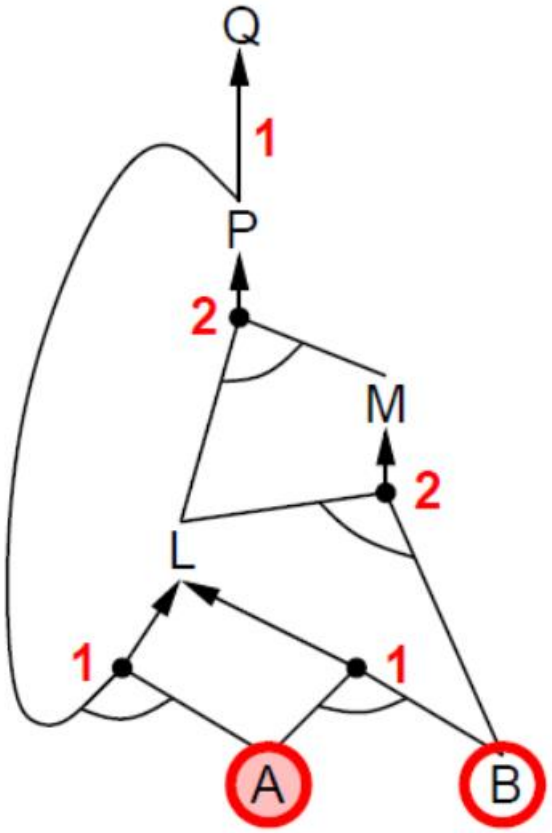
Forward-Chaining Algorithm Example

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



Forward-Chaining Algorithm 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 Algorithm Example

$P \Rightarrow Q$

$L \wedge M \Rightarrow P$

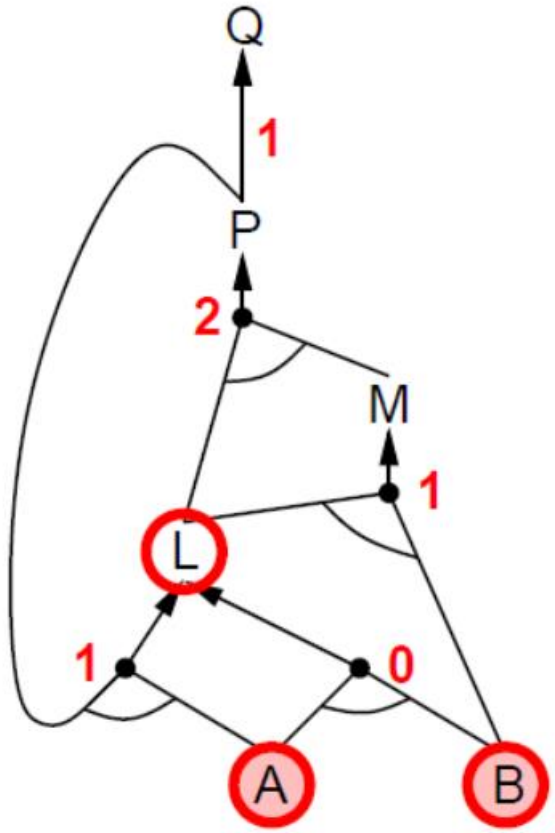
$B \wedge L \Rightarrow M$

$A \wedge P \Rightarrow L$

$A \wedge B \Rightarrow L$

A

B



Forward-Chaining Algorithm Example

$P \Rightarrow Q$

$L \wedge M \Rightarrow P$

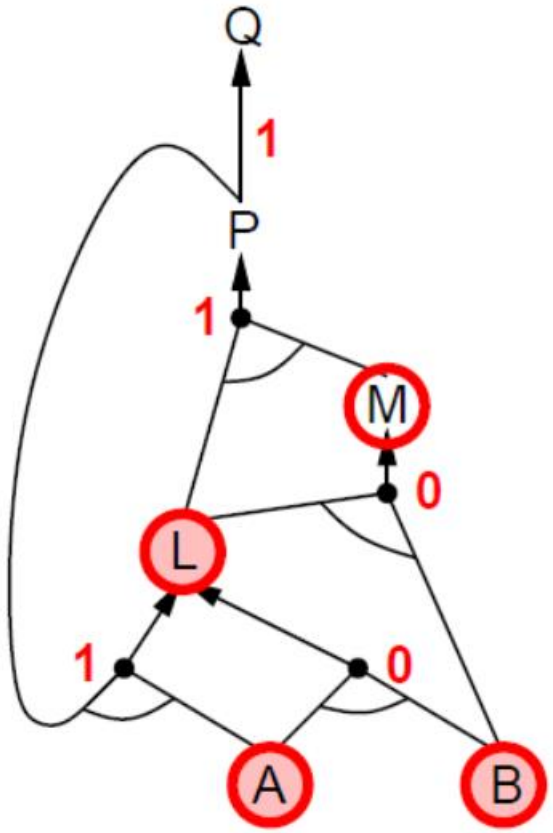
$B \wedge L \Rightarrow M$

$A \wedge P \Rightarrow L$

$A \wedge B \Rightarrow L$

A

B



Forward-Chaining Algorithm Example

$P \Rightarrow Q$

$L \wedge M \Rightarrow P$

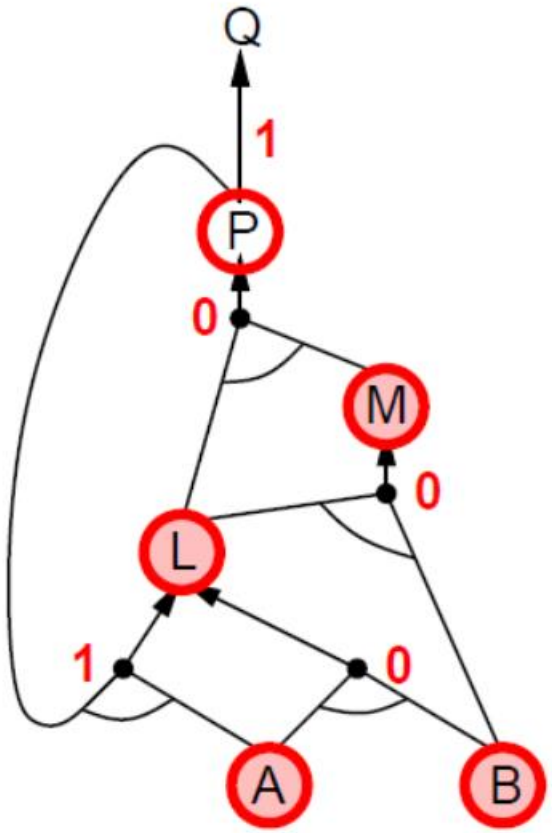
$B \wedge L \Rightarrow M$

$A \wedge P \Rightarrow L$

$A \wedge B \Rightarrow L$

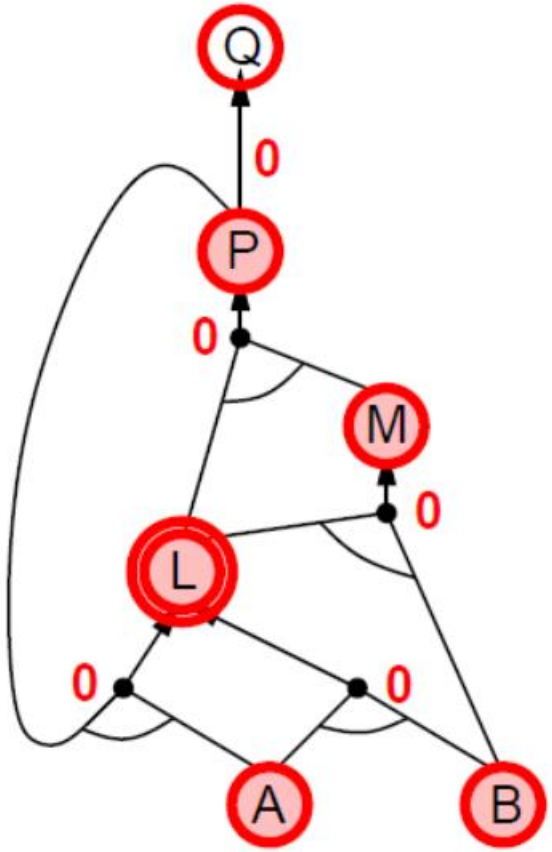
A

B



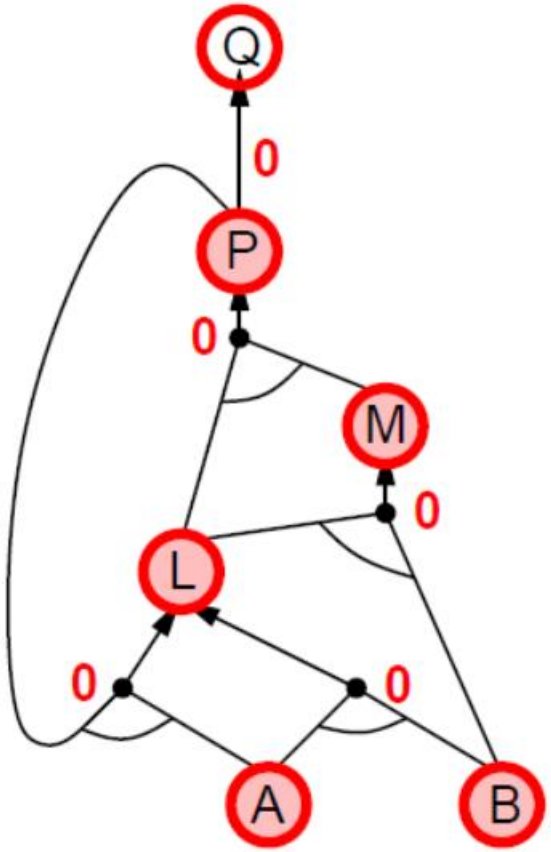
Forward-Chaining Algorithm Example

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



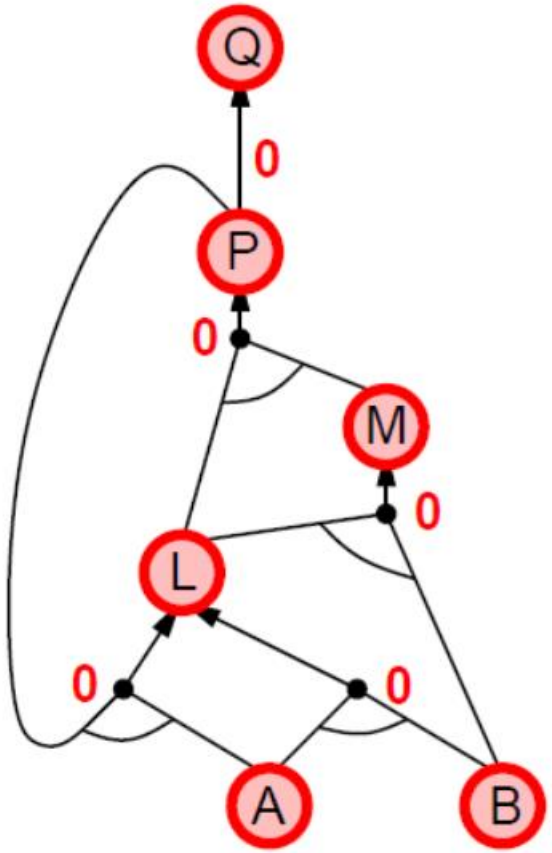
Forward-Chaining Algorithm Example

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



Forward-Chaining Algorithm 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



Proof of Completeness of FC

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 $KB \models q$, q is true in **every** model of KB, including m

General idea: construct any model of KB by sound inference, check α

Backward Chaining

- Backward chaining is a form of **goal-directed reasoning**.

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

$P \Rightarrow Q$

$L \wedge M \Rightarrow P$

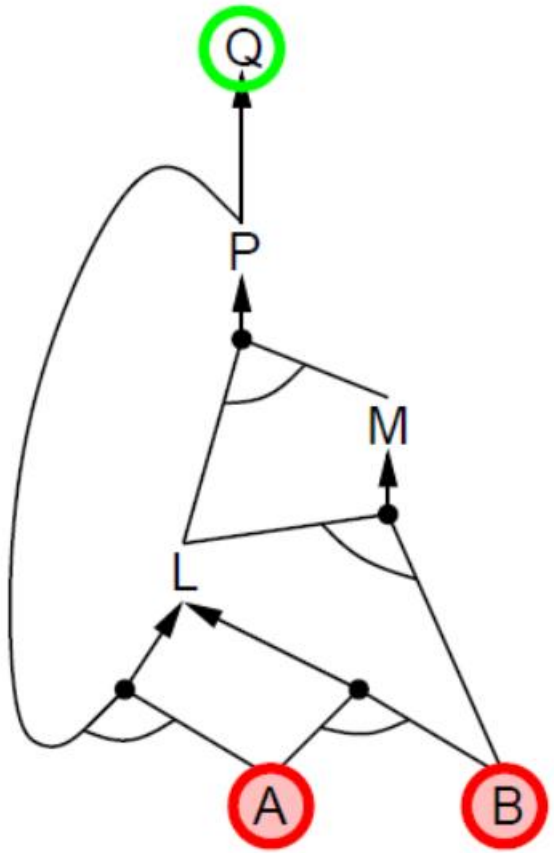
$B \wedge L \Rightarrow M$

$A \wedge P \Rightarrow L$

$A \wedge B \Rightarrow L$

A

B



Backward Chaining Example

$P \Rightarrow Q$

$L \wedge M \Rightarrow P$

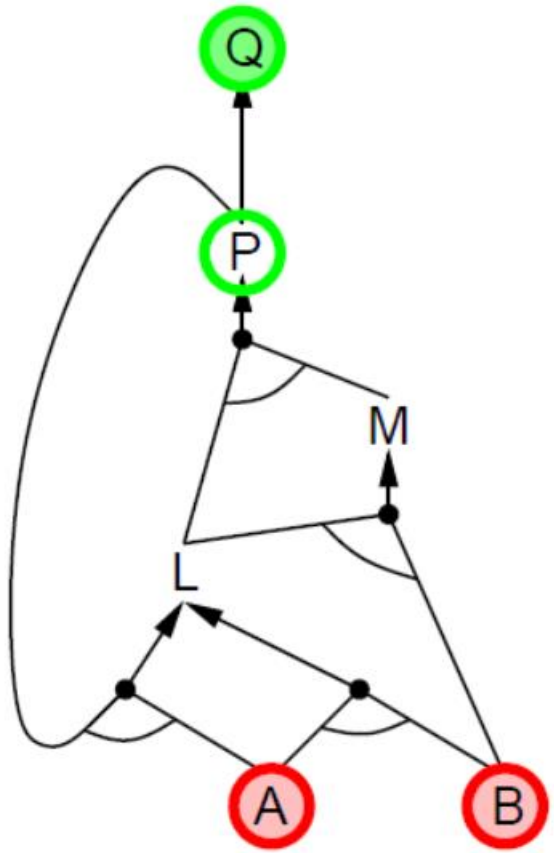
$B \wedge L \Rightarrow M$

$A \wedge P \Rightarrow L$

$A \wedge B \Rightarrow L$

A

B



Backward Chaining Example

$P \Rightarrow Q$

$L \wedge M \Rightarrow P$

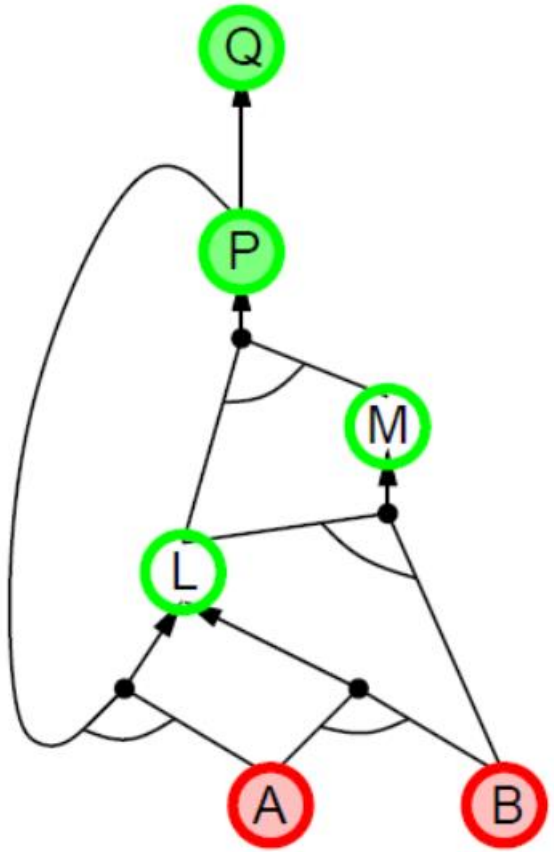
$B \wedge L \Rightarrow M$

$A \wedge P \Rightarrow L$

$A \wedge B \Rightarrow L$

A

B



Backward Chaining Example

$P \Rightarrow Q$

$L \wedge M \Rightarrow P$

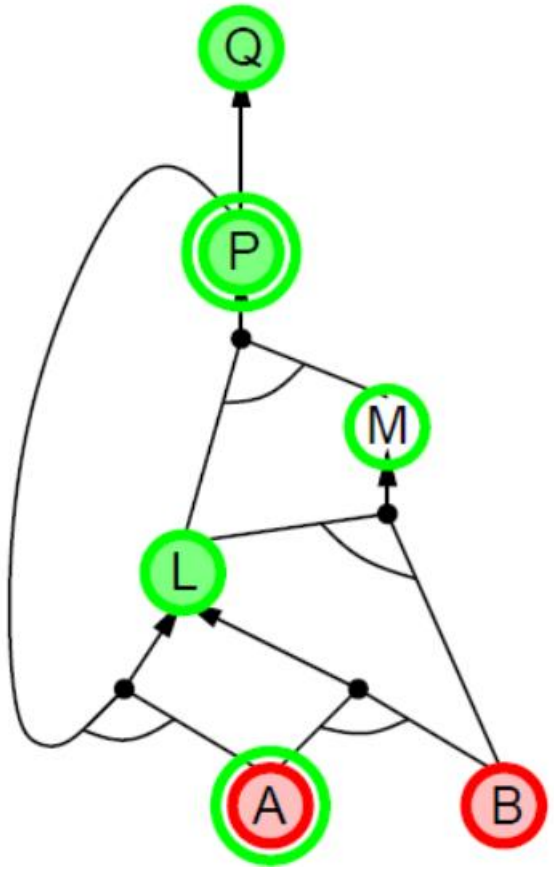
$B \wedge L \Rightarrow M$

$A \wedge P \Rightarrow L$

$A \wedge B \Rightarrow L$

A

B



Backward Chaining Example

$P \Rightarrow Q$

$L \wedge M \Rightarrow P$

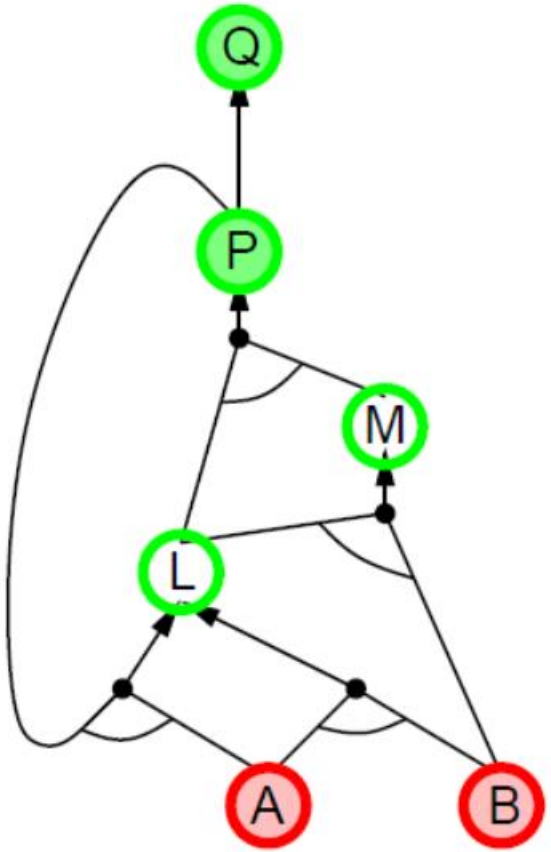
$B \wedge L \Rightarrow M$

$A \wedge P \Rightarrow L$

$A \wedge B \Rightarrow L$

A

B



Backward Chaining Example

$P \Rightarrow Q$

$L \wedge M \Rightarrow P$

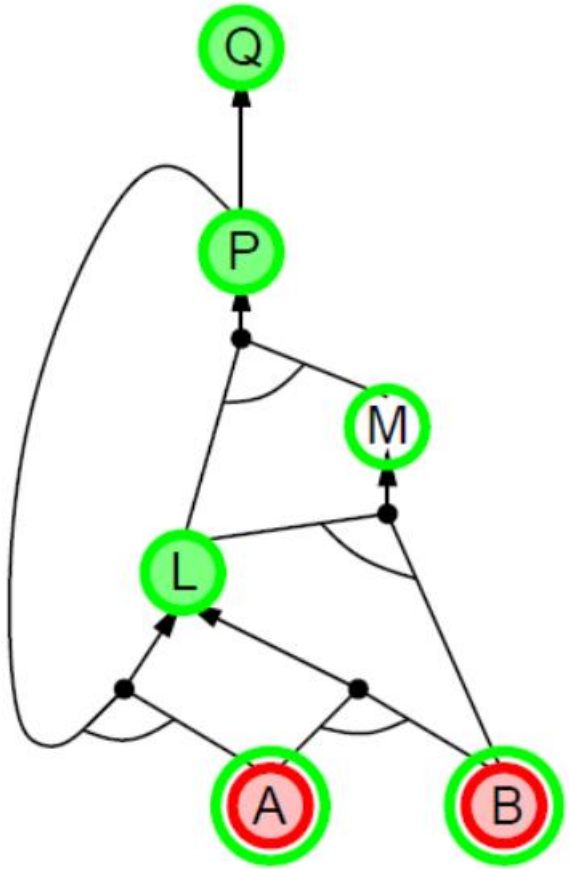
$B \wedge L \Rightarrow M$

$A \wedge P \Rightarrow L$

$A \wedge B \Rightarrow L$

A

B



Backward Chaining Example

$P \Rightarrow Q$

$L \wedge M \Rightarrow P$

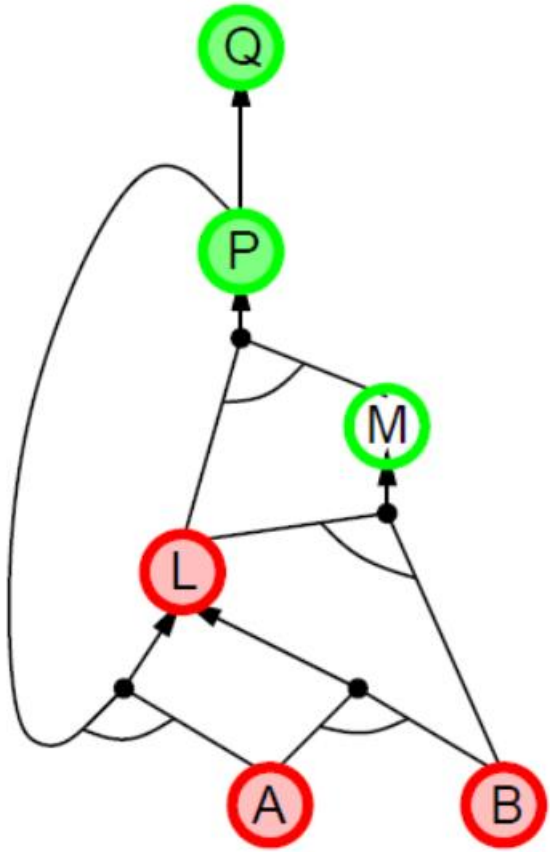
$B \wedge L \Rightarrow M$

$A \wedge P \Rightarrow L$

$A \wedge B \Rightarrow L$

A

B



Backward Chaining Example

$P \Rightarrow Q$

$L \wedge M \Rightarrow P$

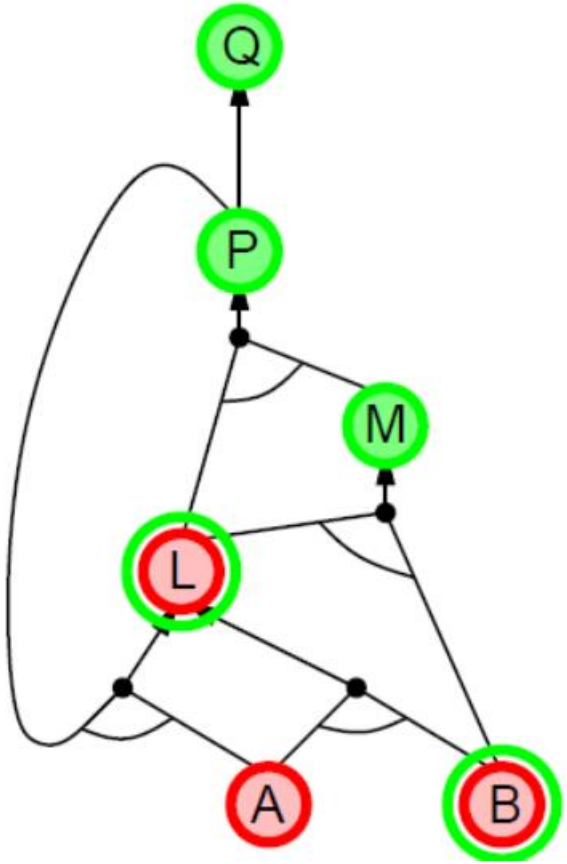
$B \wedge L \Rightarrow M$

$A \wedge P \Rightarrow L$

$A \wedge B \Rightarrow L$

A

B



Backward Chaining Example

$P \Rightarrow Q$

$L \wedge M \Rightarrow P$

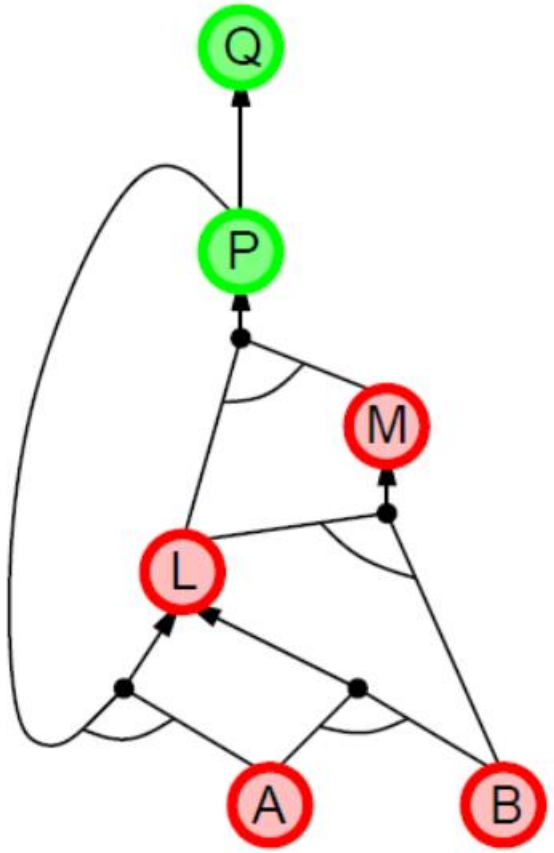
$B \wedge L \Rightarrow M$

$A \wedge P \Rightarrow L$

$A \wedge B \Rightarrow L$

A

B



Backward Chaining Example

$P \Rightarrow Q$

$L \wedge M \Rightarrow P$

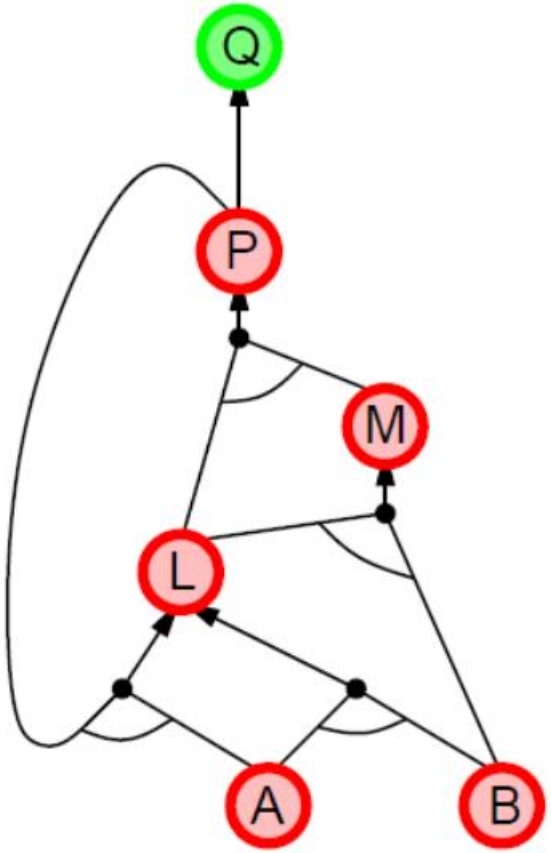
$B \wedge L \Rightarrow M$

$A \wedge P \Rightarrow L$

$A \wedge B \Rightarrow L$

A

B



Backward Chaining Example

$P \Rightarrow Q$

$L \wedge M \Rightarrow P$

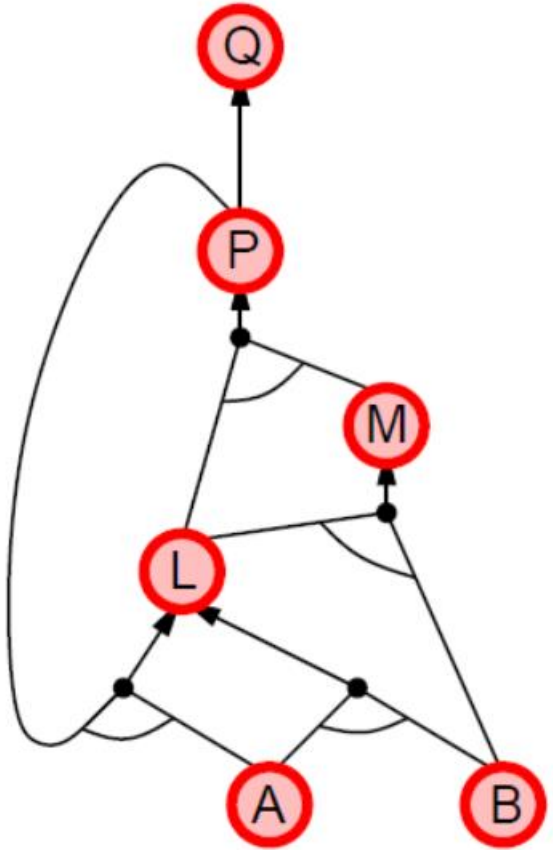
$B \wedge L \Rightarrow M$

$A \wedge P \Rightarrow L$

$A \wedge B \Rightarrow L$

A

B



Resolution Proof Example

¬P ∨ Q

¬L ∨ ¬M ∨ P

¬B ∨ ¬L ∨ M

¬A ∨ ¬P ∨ L

¬A ∨ ¬B ∨ L

A

B

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

Proof of Q

Clause1	Clause2	Resolvent
A	¬A ∨ ¬B ∨ L	¬B ∨ L
B	¬B ∨ L	L
B	¬B ∨ ¬L ∨ M	¬L ∨ M
L	¬L ∨ M	M
L	¬L ∨ ¬M ∨ P	¬M ∨ P
M	¬M ∨ P	P
P	¬P ∨ Q	Q

Forward vs. Backward Chaining

FC is data-driven, cf. automatic, unconscious processing,
e.g., object recognition, routine decisions

May do lots of work that is irrelevant to the goal

BC is goal-driven, appropriate for problem-solving,
e.g., Where are my keys? How do I get into a PhD program?

Complexity of BC can be much less than linear in size of KB

Summary

- 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
  - soundness: 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