Logical Agents

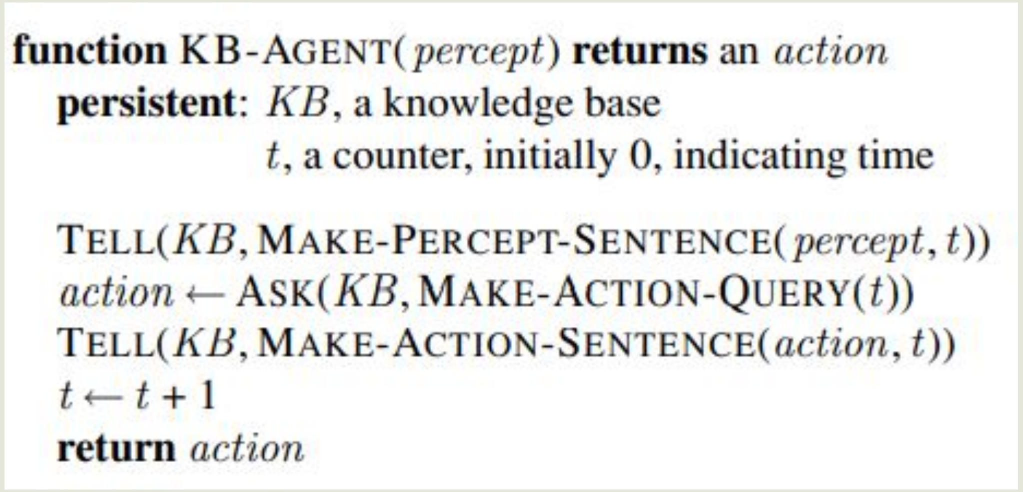
Shiraz Khurana

KNOWLEDGE-BASED AGENTS

- A knowledge base is a set of sentences. Each sentence is expressed in a language called a knowledge representation language and represents some assertion about the world.
- when the sentence is taken as given without being derived from other sentences is called as **axioms.**
- There must be a way to add new sentences to the knowledge base and a way to query what is known. The standard names for these operations are TELL and ASK, respectively.
- Both the operations may involve inference—that is, deriving new sentences from old.

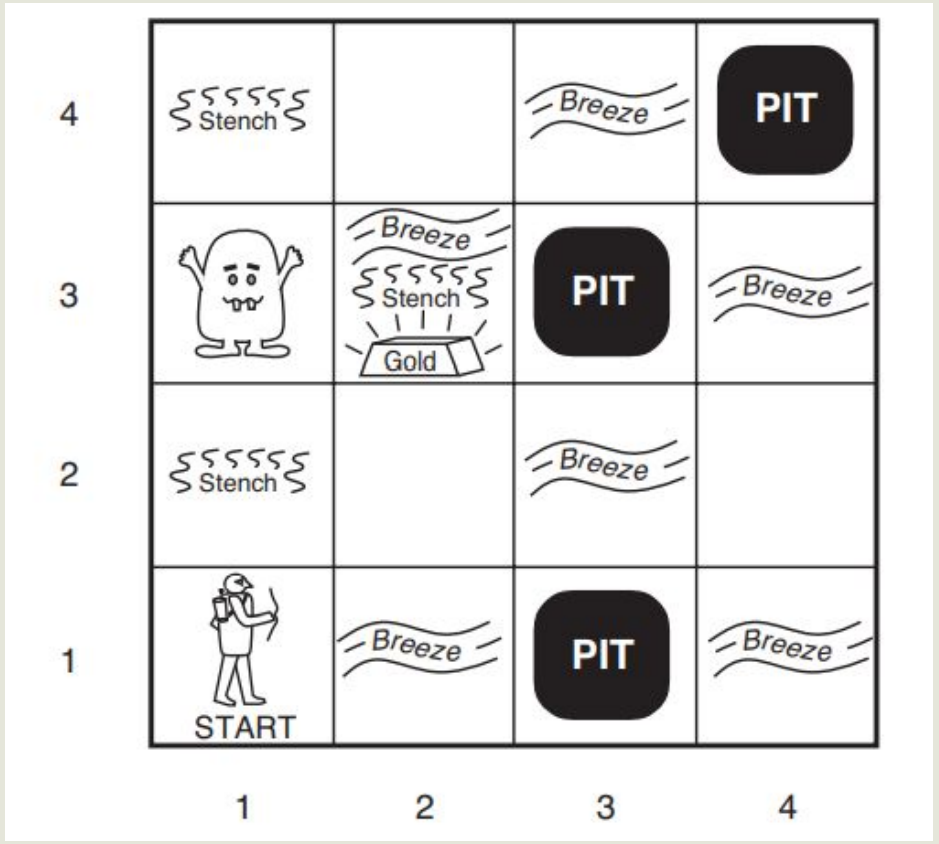
KNOWLEDGE-BASED AGENTS

- Each time the agent program is called, it does three things.
  - It TELLs the knowledge base what it perceives.
  - It ASKs the knowledge base what action it should perform. In the process of answering this query, extensive reasoning may be done about the current state of the world, about the outcomes of possible action sequences.
  - The agent program TELLs the knowledge base which action was chosen, and the agent executes the 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.



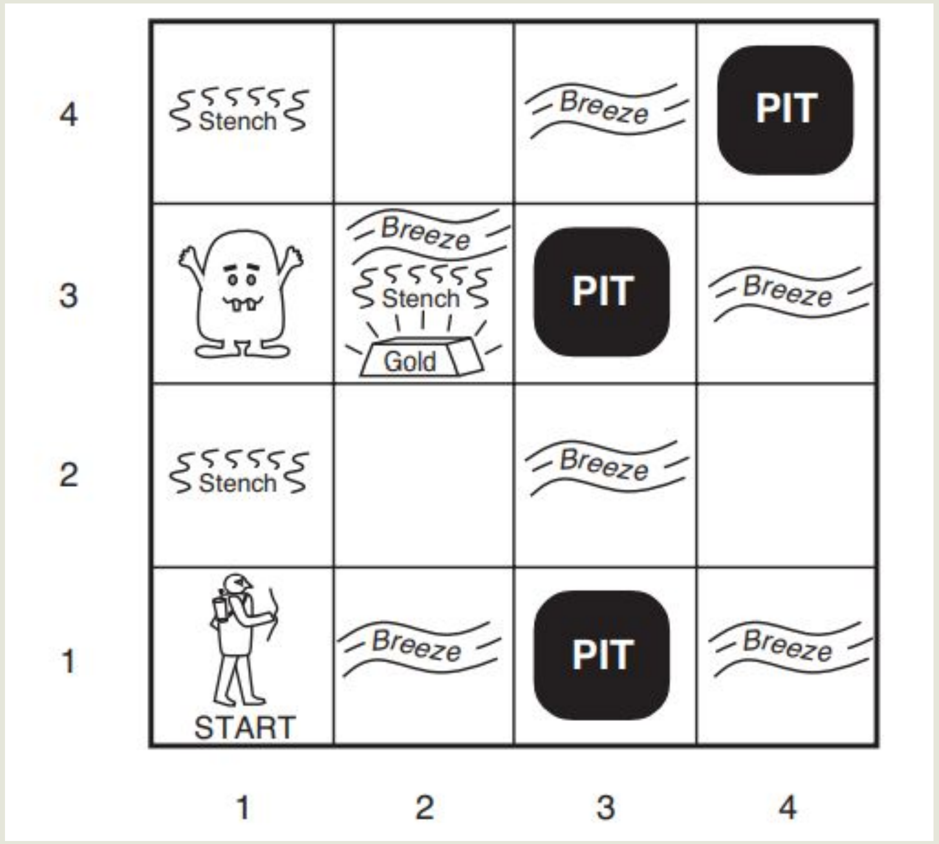
```
function KB-AGENT(percept) returns an action
  persistent: KB, a knowledge base
              t, a counter, initially 0, indicating time

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

THE WUMPUS WORLD



- Performance Measure: +1000 for climbing out of the cave with the gold, –1000 for falling into a pit or being eaten by the wumpus, –1 for each action taken and –10 for using up the arrow.
- Environment: A 4 × 4 grid of rooms. The agent always starts in the square labeled [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.
- Actuators: The agent can move Forward, Turn Left by 90◦, or Turn Right by 90◦. The agent dies a miserable death if it enters a square containing a pit or a live Wumpus. If an agent tries to move forward and bumps into a wall, then the agent does not move. The action Grab can be used to pick up the gold if it is in the same square as the agent.

THE WUMPUS WORLD

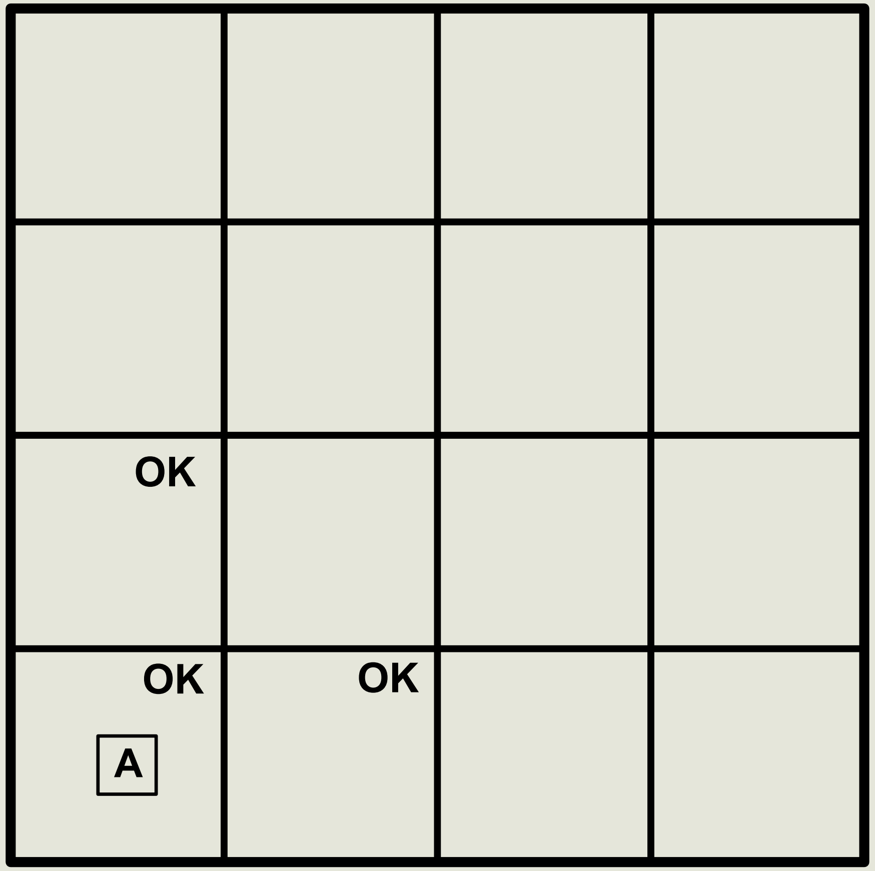


- Actuators: The action Shoot can be used to fire an arrow in a straight line in the direction the agent is facing. The agent has only one arrow, so only the first Shoot action has any effect. Finally, the action Climb can be used to climb out of the cave, but only from square [1,1].
- Sensors: The agent has five sensors, each of which gives a single bit of information: –
  - In the square containing the wumpus and in the directly (not diagonally) adjacent squares, the agent will perceive a Stench.
  - In the squares directly adjacent to a pit, the agent will perceive a Breeze.
  - In the square where the gold is, the agent will perceive a Glitter.
  - When an agent walks into a wall, it will perceive a Bump.
  - When the wumpus is killed, it emits a woeful Scream that can be perceived anywhere in the cave.

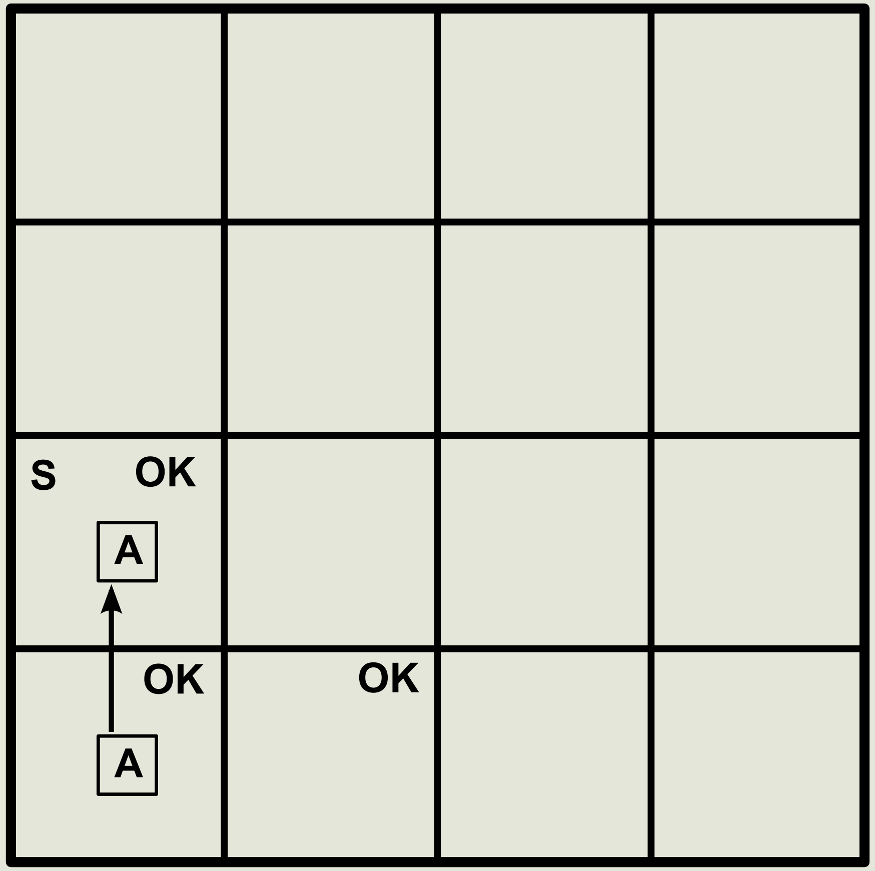
Characterization of Wumpus World

- The percepts will be given to the agent program in the form of a list of five symbols; for example, if there is a stench and a breeze, but no glitter, bump, or scream, the agent program will get [Stench, Breeze, None, None, None].
- Partially Observable
- Deterministic
- Sequential
- Static
- Discrete
- Single Agent

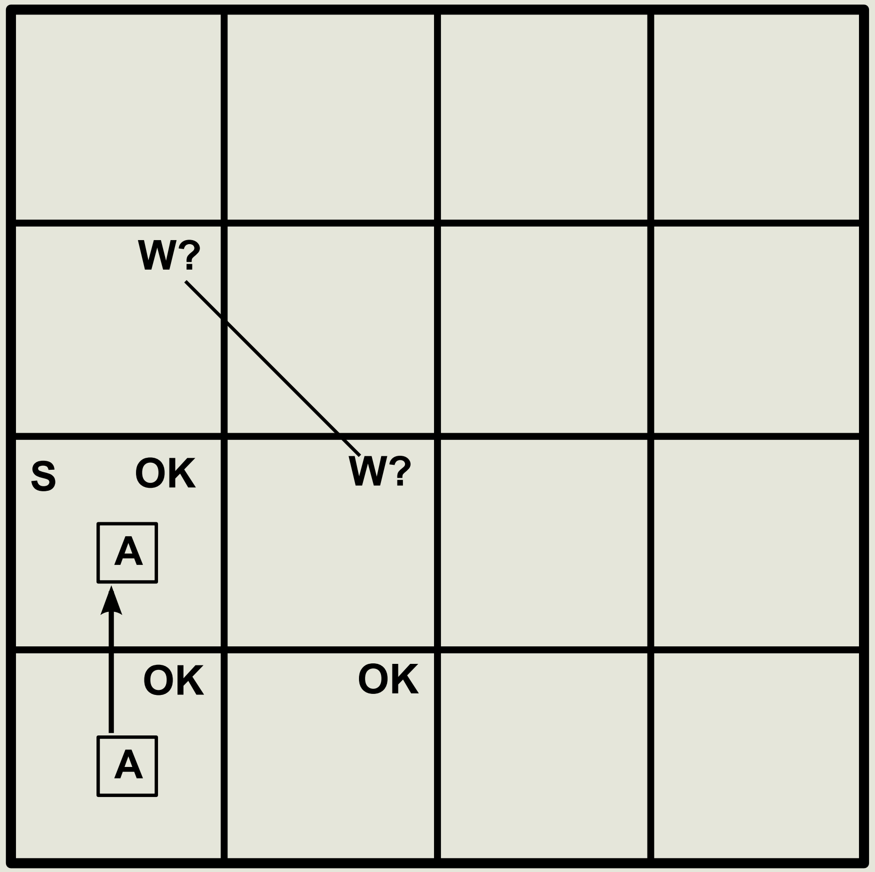
Exploring a Wumpus World



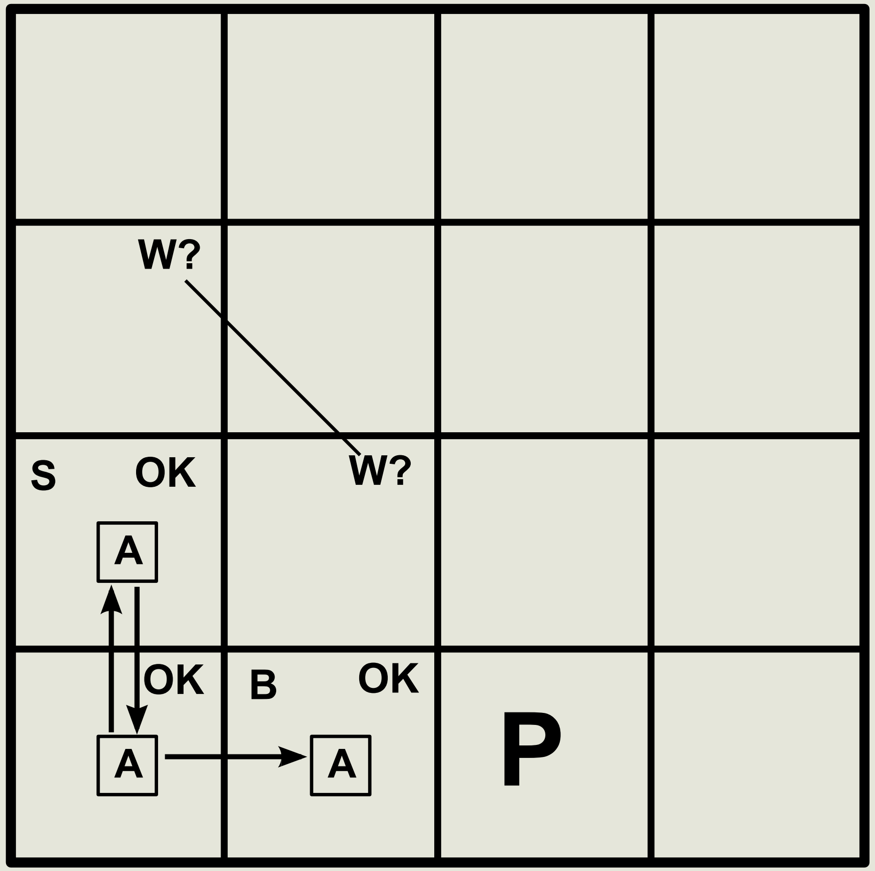
Exploring a Wumpus World



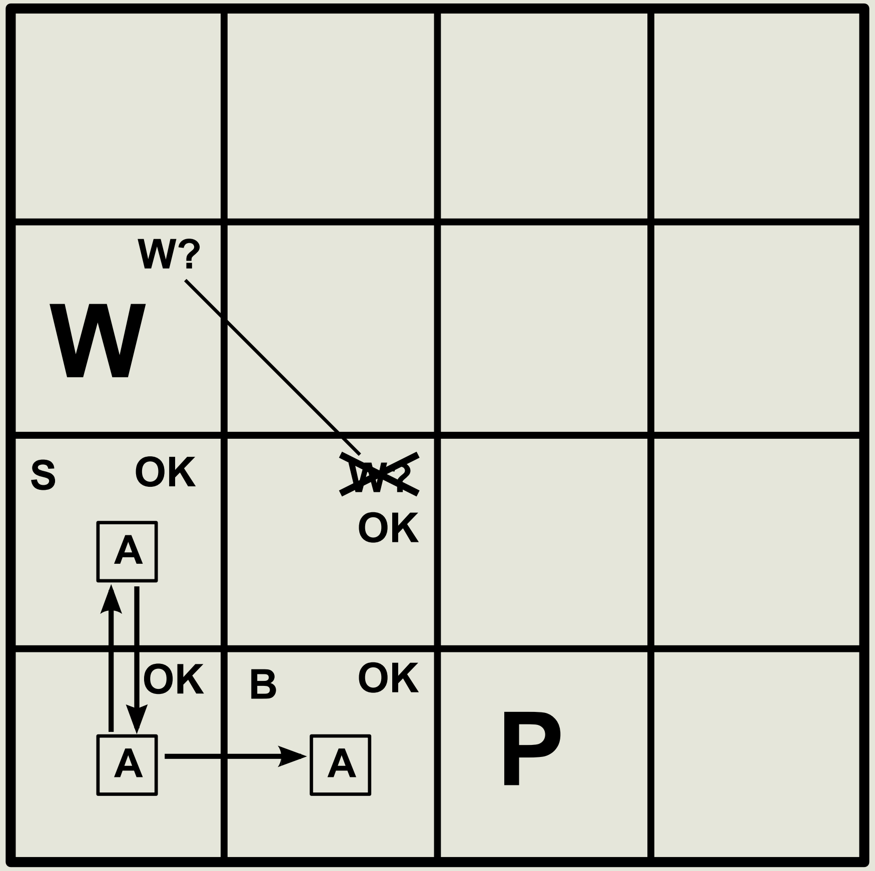
Exploring a Wumpus World



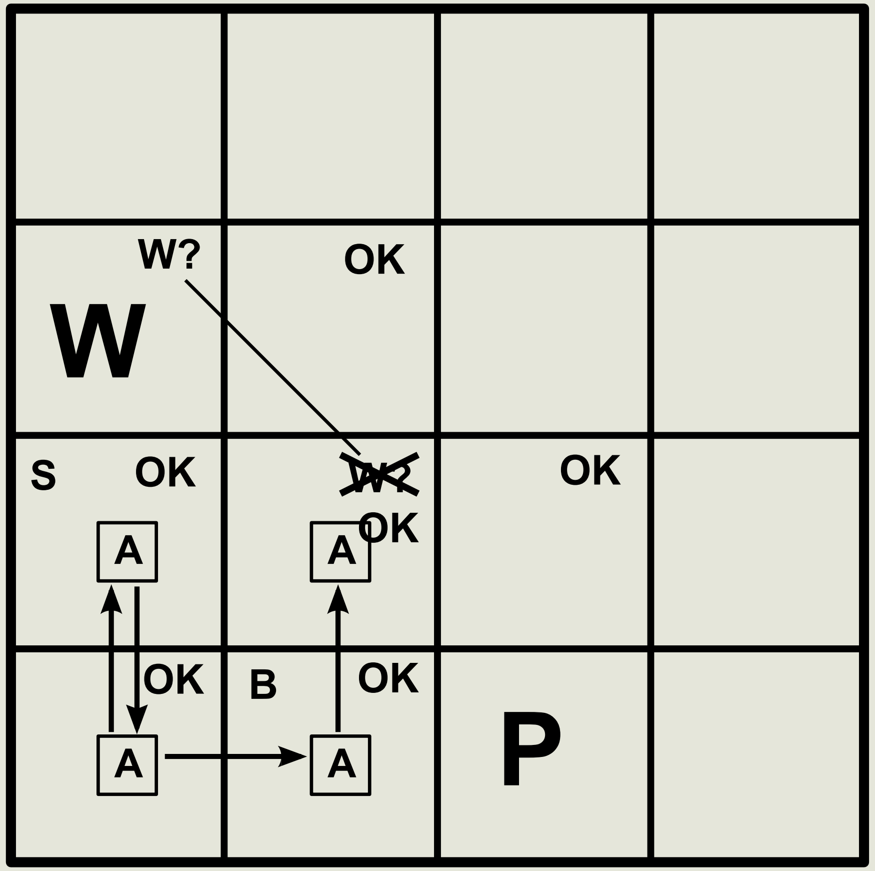
Exploring a Wumpus World



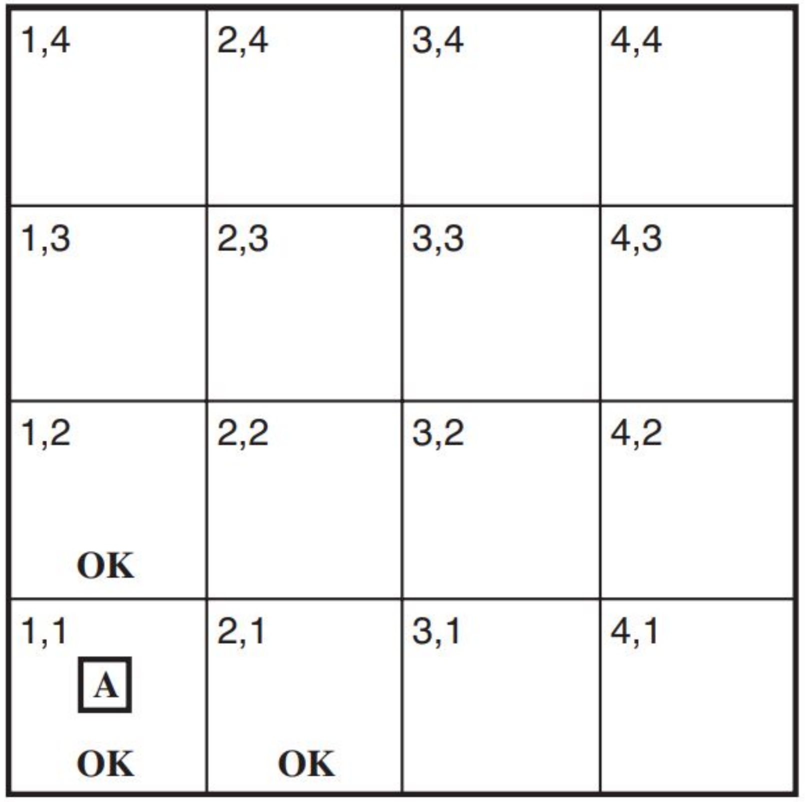
Exploring a Wumpus World



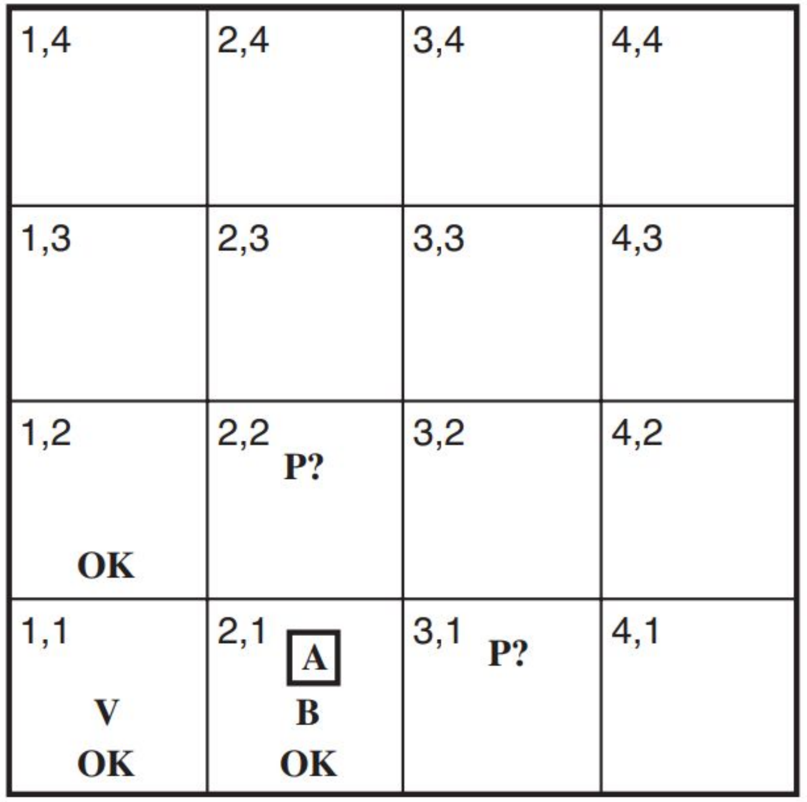
Exploring a Wumpus World



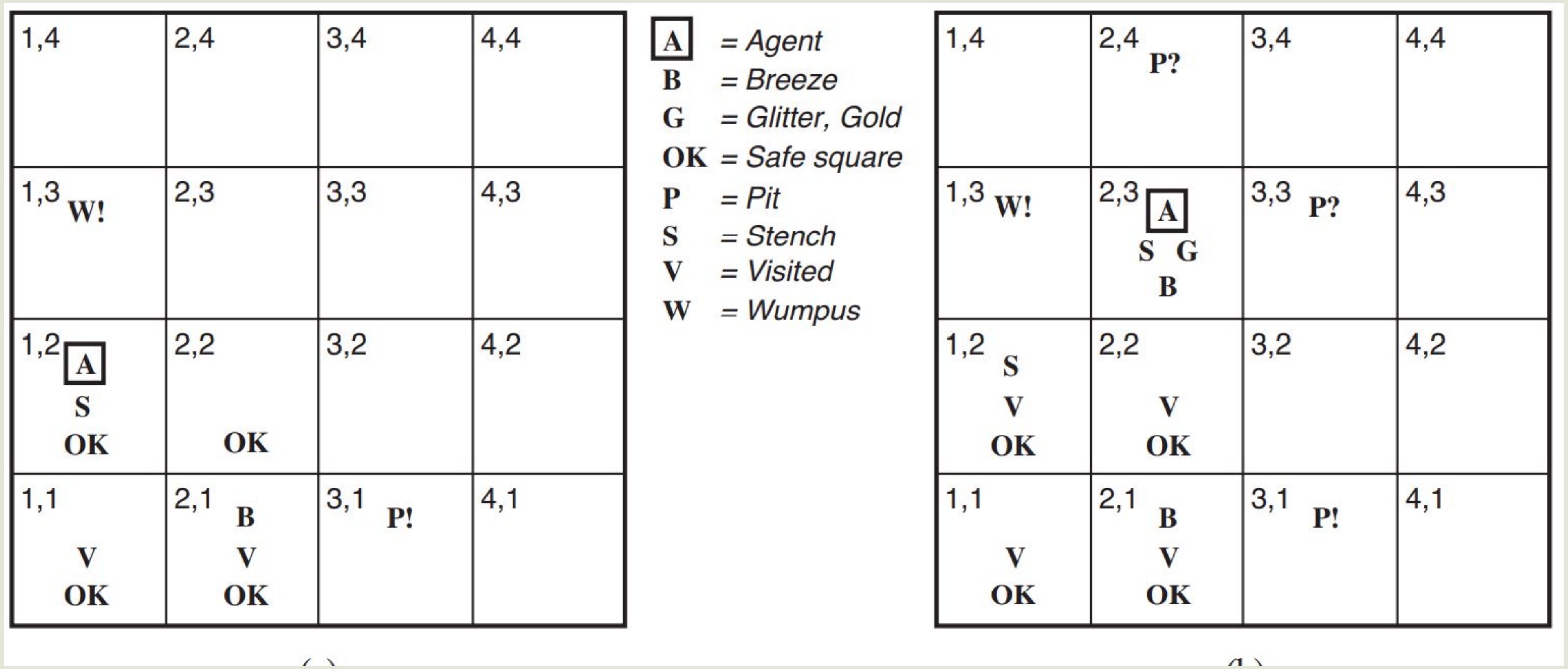
Exploring a Wumpus World



A	= *Agent*
B	= *Breeze*
G	= *Glitter, Gold*
OK	= *Safe square*
P	= *Pit*
S	= *Stench*
V	= *Visited*
W	= *Wumpus*



Exploring a Wumpus World



LOGIC

- The sentences are expressed according to the syntax of the representation language, which specifies all the sentences that are well formed.
- Logics are formal languages for representing information such that conclusions can be drawn. Syntax defines the sentences in the language.
- A logic must also define the semantics or meaning of sentences. The semantics defines the truth of each sentence with respect to each possible 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 in a world where $x = 7$; $y = 1$
- $x + 2 \geq y$ is false in a world where $x = 0$; $y = 6$

Models

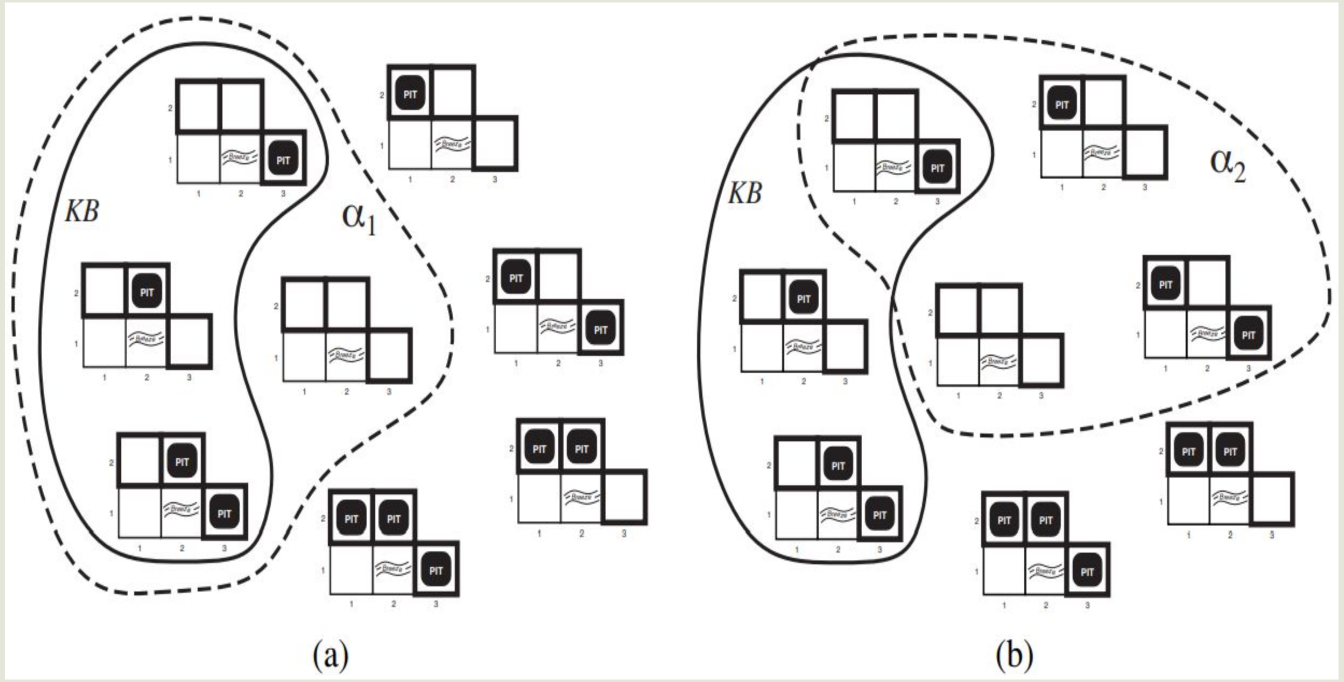
- We use **Models** in logics: models are mathematical abstractions, each of which simply fixes the truth or falsehood of every relevant sentence.
- for example, having x men and y women sitting at a table playing bridge, and the sentence $x + y = 4$ is true when there are four people in total.
- the possible models are just all possible assignments of real numbers to the variables x and y. Each such assignment fixes the truth of any sentence of arithmetic whose variables are x and y.
- If a sentence α is true in model m, we say that m satisfies α or sometimes m is a model of α. We use the notation $M(\alpha)$ to mean the set of all models of α.

Entailment

- Entailment means that one thing follows from another: $\alpha \models \beta$
- mean that the sentence α entails the sentence β.
- The formal definition of entailment is this: $\alpha \models \beta$ if and only if, in every model in which α is true, β is also true.
- $\alpha \models \beta$ if and only if $M(\alpha) \subseteq M(\beta)$.
- 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

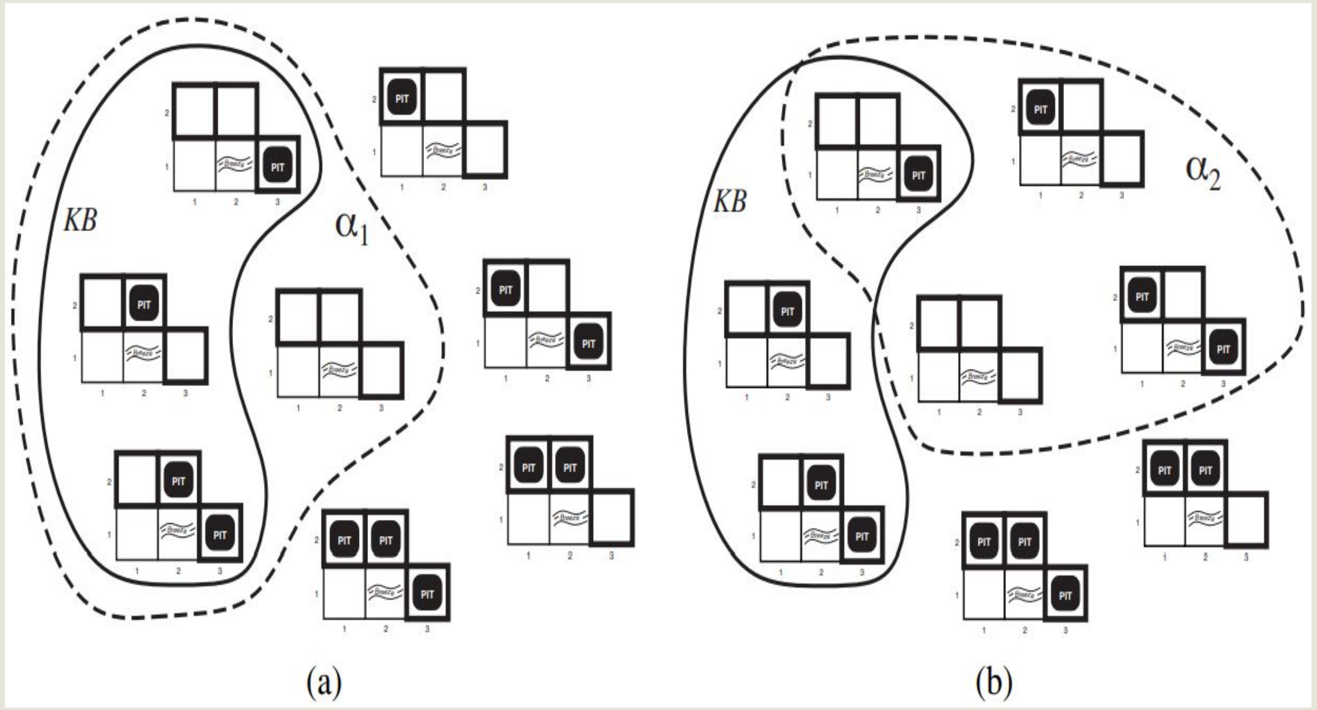
Entailment in Wumpus World

- The agent has detected nothing in [1,1] and a breeze in [2,1]. These percepts, combined with the agent’s knowledge of the rules of the wumpus world, constitute the KB.
- The agent is interested in whether the adjacent squares [1,2], [2,2], and [3,1] contain pits. Each of the three squares might or might not contain a pit, there are 8 possible models.
- The KB is set of sentences or as a single sentence that asserts all the individual sentences. **The KB is false in models that contradict what the agent knows.**
- e.g., the KB is false in any model in which [1,2] contains a pit, because there is no breeze in [1,1]



Detailed Example

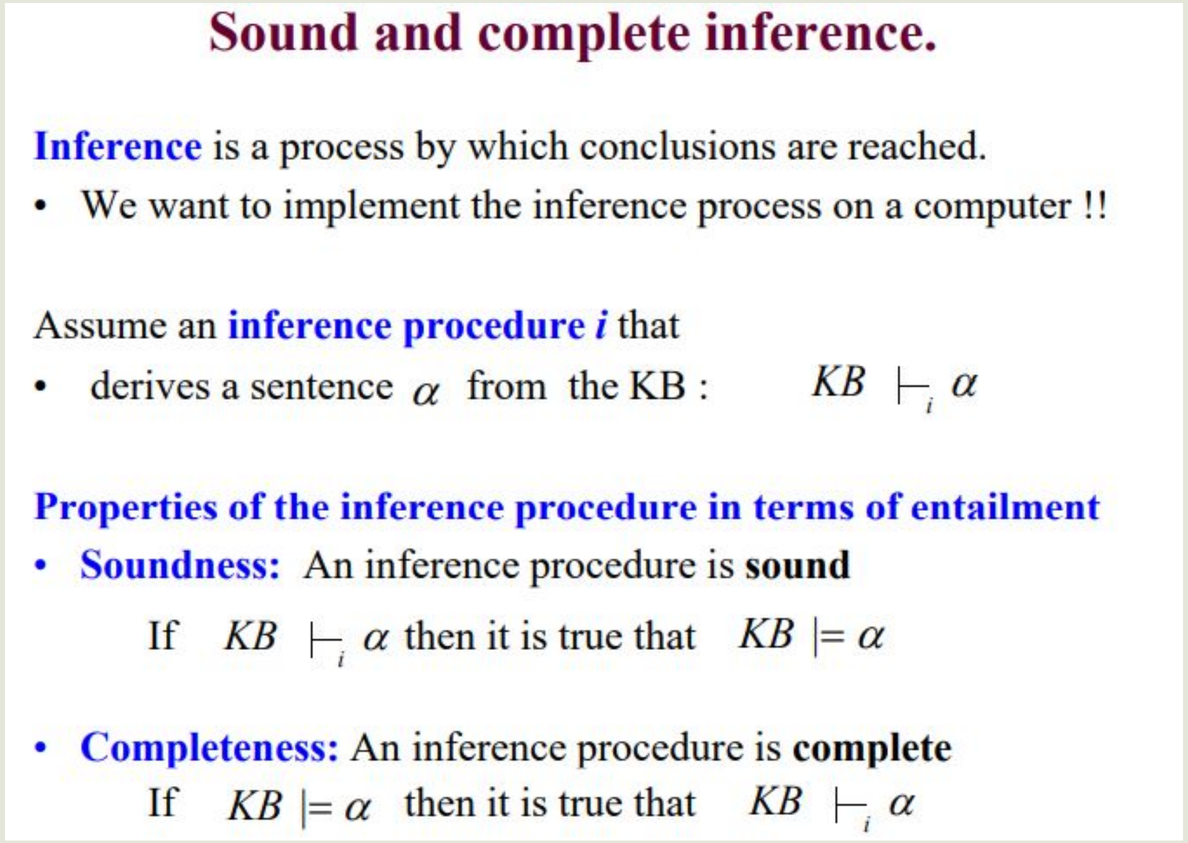
- α1 = "There is no pit in [1,2]."
- α2 = "There is no pit in [2,2]."
- in every model in which KB is true, α1 is also true.
- Hence, KB $\models$ α1: there is no pit in [1,2].
- it enumerates all possible models to check that α is true in all models in which KB is true, that is, that M(KB) ⊆ M(α).
- in some models in which KB is true, α2 is false.
- Hence, KB $\not\models$ α2: the agent cannot conclude that there is no pit in [2,2].



Soundness & Completeness

- Consider the set of all consequences of KB as a haystack and of α as a needle. Entailment is like the needle being in the haystack; inference is like finding it.
- Inference is a process by which conclusions are reached. We want to implement the inference process on a computer !!
- Assume an inference procedure i that derives a sentence α from the KB : $KB \vdash_i \alpha$
- "α is derived from KB by i" or "i derives α from KB.
- An inference algorithm that derives only entailed sentences is called **sound** or **truth preserving**.
- an inference algorithm is **complete** if it can derive any sentence that is entailed by KB.
- if KB is true in the real world, then any sentence α derived from KB by a sound inference procedure is also true in the real world.

Soundness & Completeness



Sound and complete inference.

Inference is a process by which conclusions are reached.

- We want to implement the inference process on a computer !!

Assume an **inference procedure *i*** that

- derives a sentence α from the KB : $KB \vdash_i \alpha$

Properties of the inference procedure in terms of entailment

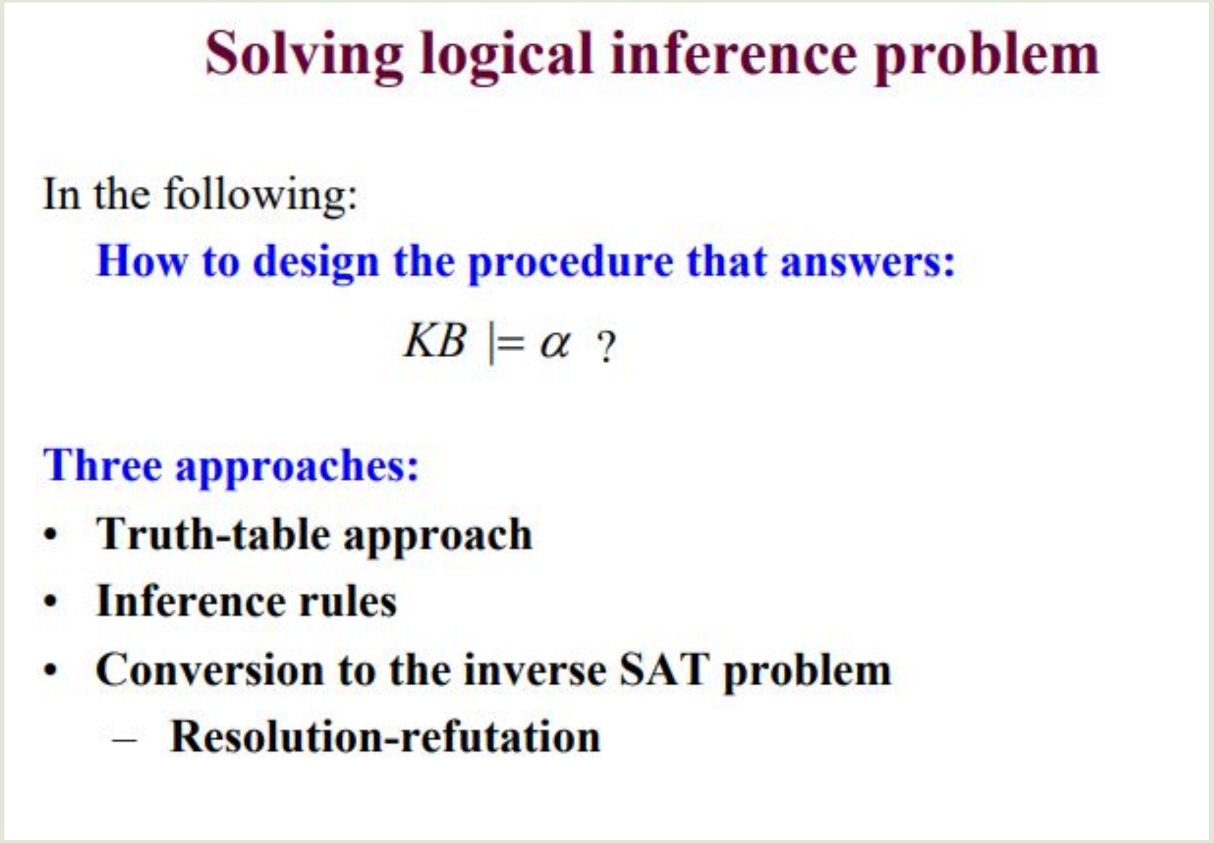
- **Soundness:** An inference procedure is **sound**

  If $KB \vdash_i \alpha$ then it is true that $KB \models \alpha$

- **Completeness:** An inference procedure is **complete**

  If $KB \models \alpha$ then it is true that $KB \vdash_i \alpha$

Solve Inference Problems



Solving logical inference problem

In the following:

How to design the procedure that answers:

$$KB \models \alpha \ ?$$

Three approaches:

- **Truth-table approach**
- **Inference rules**
- **Conversion to the inverse SAT problem**
  - **Resolution-refutation**

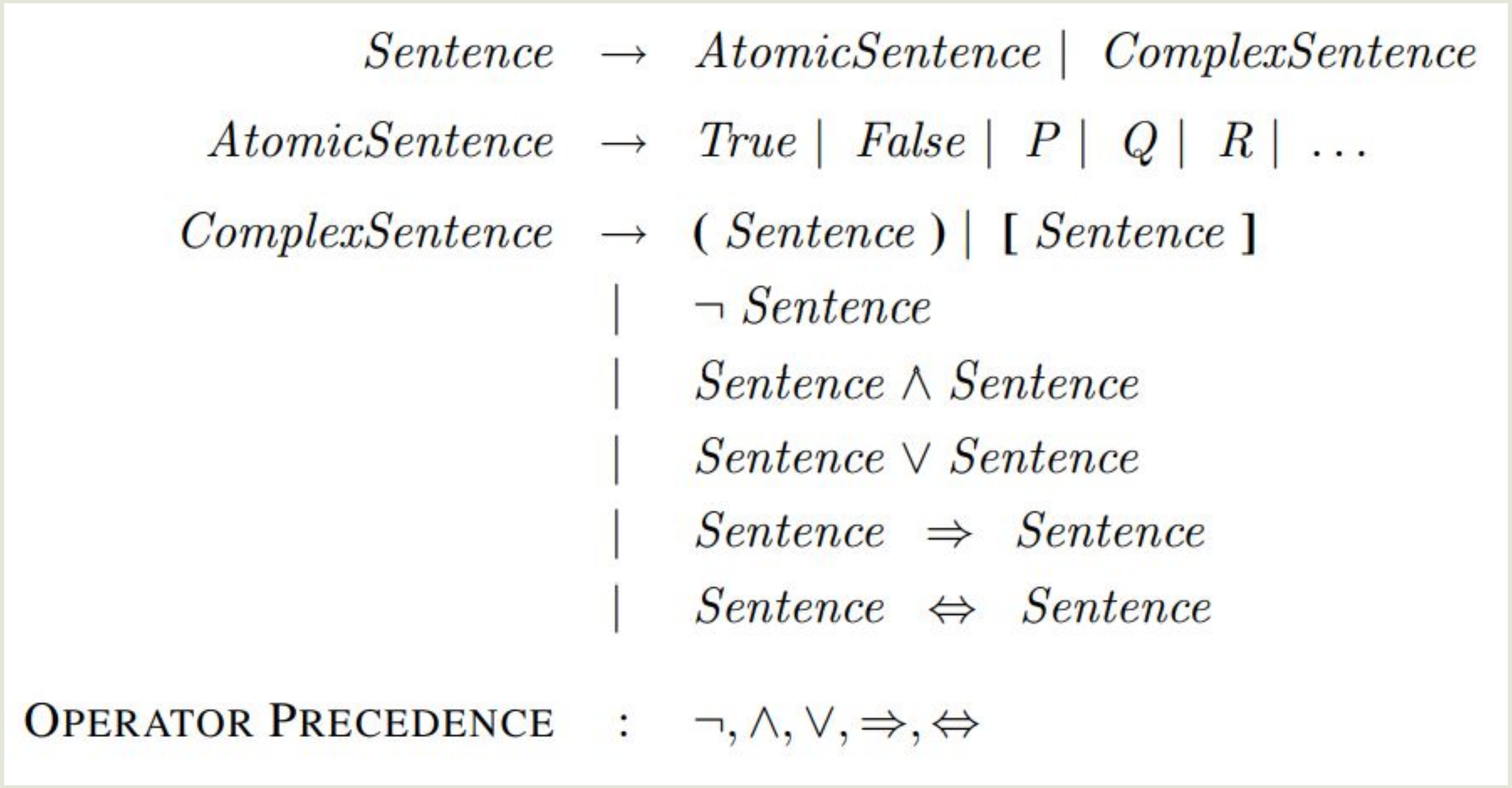
PROPOSITIONAL LOGIC

- The syntax of propositional logic defines the allowable sentences. The atomic sentences consist of a single proposition symbol. Each such symbol stands for a proposition that can be true or false.
- The symbols that start with an uppercase letter and may contain other letters or subscripts, for example: P, Q, R, $W_{1,3}$ and North.
- Complex sentences are constructed from simpler sentences, using parentheses and logical connectives. There are five connectives in common use:
- $\neg$ (not). A sentence such as $\neg W_{1,3}$ is called the negation of $W_{1,3}$. A literal is either an atomic sentence (a positive literal) or a negated atomic sentence (a negative literal).
- $\land$ (and). A sentence whose main connective is $\land$, such as $W_{1,3} \land P_{3,1}$, is called a conjunction; its parts are the conjuncts.

PROPOSITIONAL LOGIC

- $\lor$ (or). A sentence using $\lor$, such as $(W_{1,3} \land P_{3,1}) \lor W_{2,2}$, is a disjunction of the disjuncts $(W_{1,3} \land P_{3,1})$ and $W_{2,2}$.
- $\Rightarrow$ (implies). A sentence such as $(W_{1,3} \land P_{3,1}) \Rightarrow \neg W_{2,2}$ is called an implication (or conditional). Its premise or antecedent is $(W1,3 \land P3,1)$, and its conclusion or consequent is $\neg W2,2$.
- $\Leftrightarrow$ (if and only if). The sentence $W_{1,3} \Leftrightarrow \neg W_{2,2}$ is a bi-conditional.

BNF (Backus–Naur Form) grammar of sentences in propositional logic

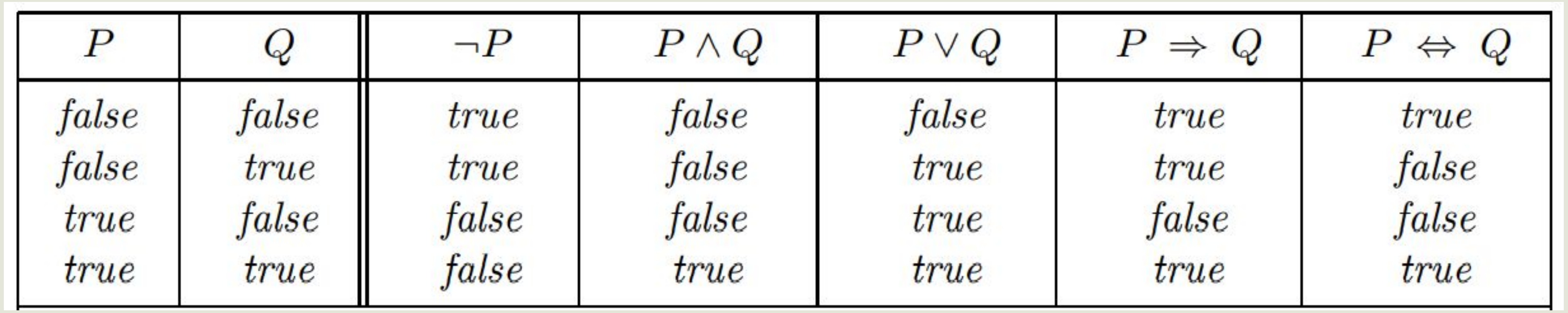


$$
\begin{array}{rcl}
\mathit{Sentence} & \rightarrow & \mathit{AtomicSentence} \mid \mathit{ComplexSentence} \\
\mathit{AtomicSentence} & \rightarrow & \mathit{True} \mid \mathit{False} \mid P \mid Q \mid R \mid \ldots \\
\mathit{ComplexSentence} & \rightarrow & (\ \mathit{Sentence}\ ) \mid [\ \mathit{Sentence}\ ] \\
 & \mid & \neg\ \mathit{Sentence} \\
 & \mid & \mathit{Sentence} \land \mathit{Sentence} \\
 & \mid & \mathit{Sentence} \lor \mathit{Sentence} \\
 & \mid & \mathit{Sentence} \ \Rightarrow \ \mathit{Sentence} \\
 & \mid & \mathit{Sentence} \ \Leftrightarrow \ \mathit{Sentence} \\
 & & \\
\textsc{Operator Precedence} & : & \neg, \land, \lor, \Rightarrow, \Leftrightarrow
\end{array}
$$

Semantics of Propositional Logic

- The semantics defines the rules for determining the truth of a sentence with respect to a particular model.
- if the sentences in the knowledge base make use of the proposition symbols $P_{1,2}$, $P_{2,2}$, and $P_{3,1}$, then one possible model is m1 = $\{P_{1,2} = false, P_{2,2} = false, P_{3,1} = true\}$.
- For complex sentences, we have five rules, which hold for any sub-sentences P and Q in any model m:
  - $\neg P$ is true iff P is false in m.
  - $P \wedge Q$ is true iff both P and Q are true in m.
  - $P \vee Q$ is true iff either P or Q is true in m.
  - $P \Rightarrow Q$ is true unless P is true and Q is false in m.
  - $P \Leftrightarrow Q$ is true iff P and Q are both true or both false in m.

Truth Table



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*

A simple knowledge base

- $P_{x,y}$ 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]
- each sentence is labeled as R_i

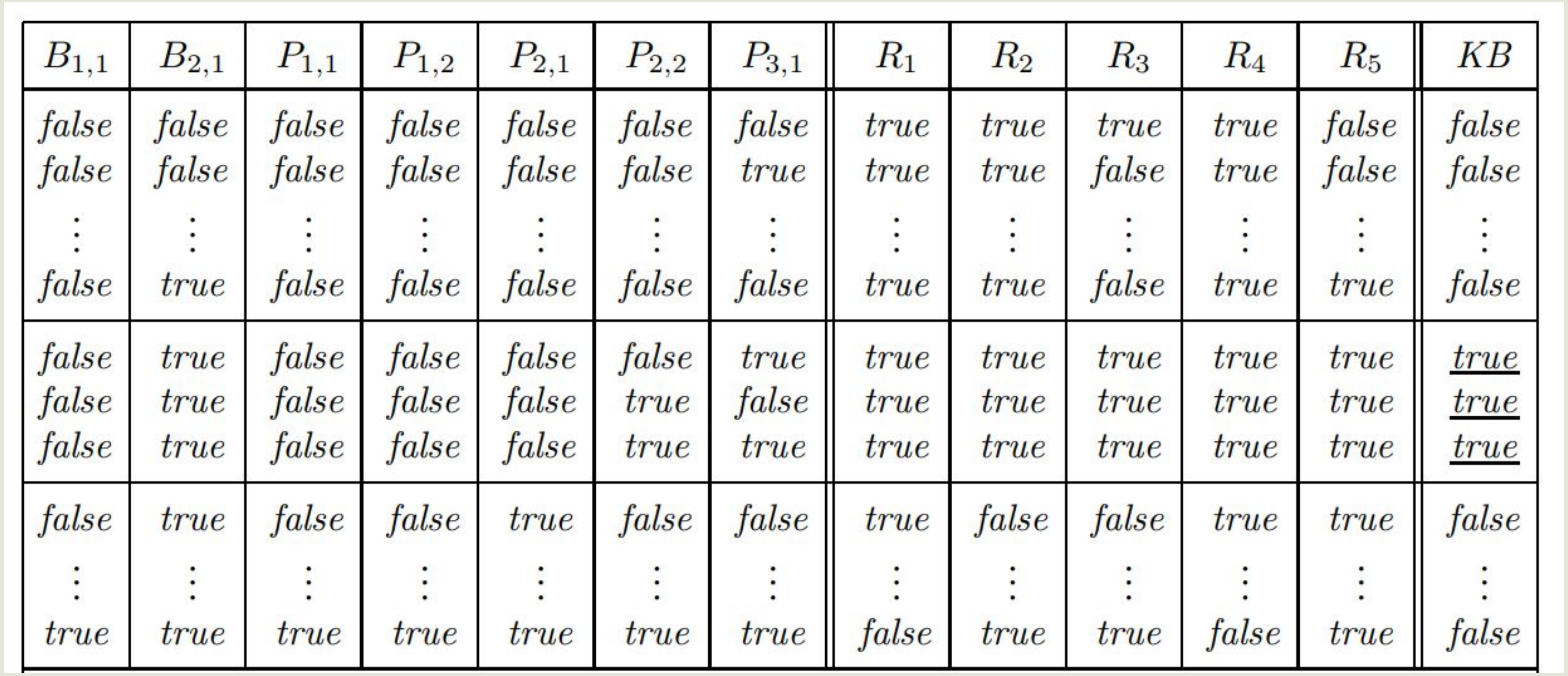
knowledge base wumpus world

- There is no pit in [1,1]: R1 : $\neg P_{1,1}$.
- A square is breezy if and only if there is a pit in a neighboring square. This has to be stated for each square; for now, we include just the relevant squares:
- R2 : $B_{1,1} \Leftrightarrow (P_{1,2} \lor P_{2,1})$.
- R3 : $B_{2,1} \Leftrightarrow (P_{1,1} \lor P_{2,2} \lor P_{3,1})$.
- Now we include the breeze percepts for the first two squares visited in the specific world the agent is in.
- R4 : $\neg B_{1,1}$.
- R5 : B2,1 .

A simple inference procedure (Propositional Logic)

- First algorithm for inference is model checking that 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. Returning to our wumpus-world example, the relevant proposition symbols are $B_{1,1}$, $B_{2,1}$, $P_{1,1}$, $P_{1,2}$, $P_{2,1}$, $P_{2,2}$, and $P_{3,1}$.
- The algorithm is sound because it implements directly the definition of entailment, and complete because it works for any KB and α and always terminates—there are only finitely many models to examine.
- every known inference algorithm for propositional logic has a worst-case complexity that is exponential in the size of the input.

Truth Table for Model Based Approach

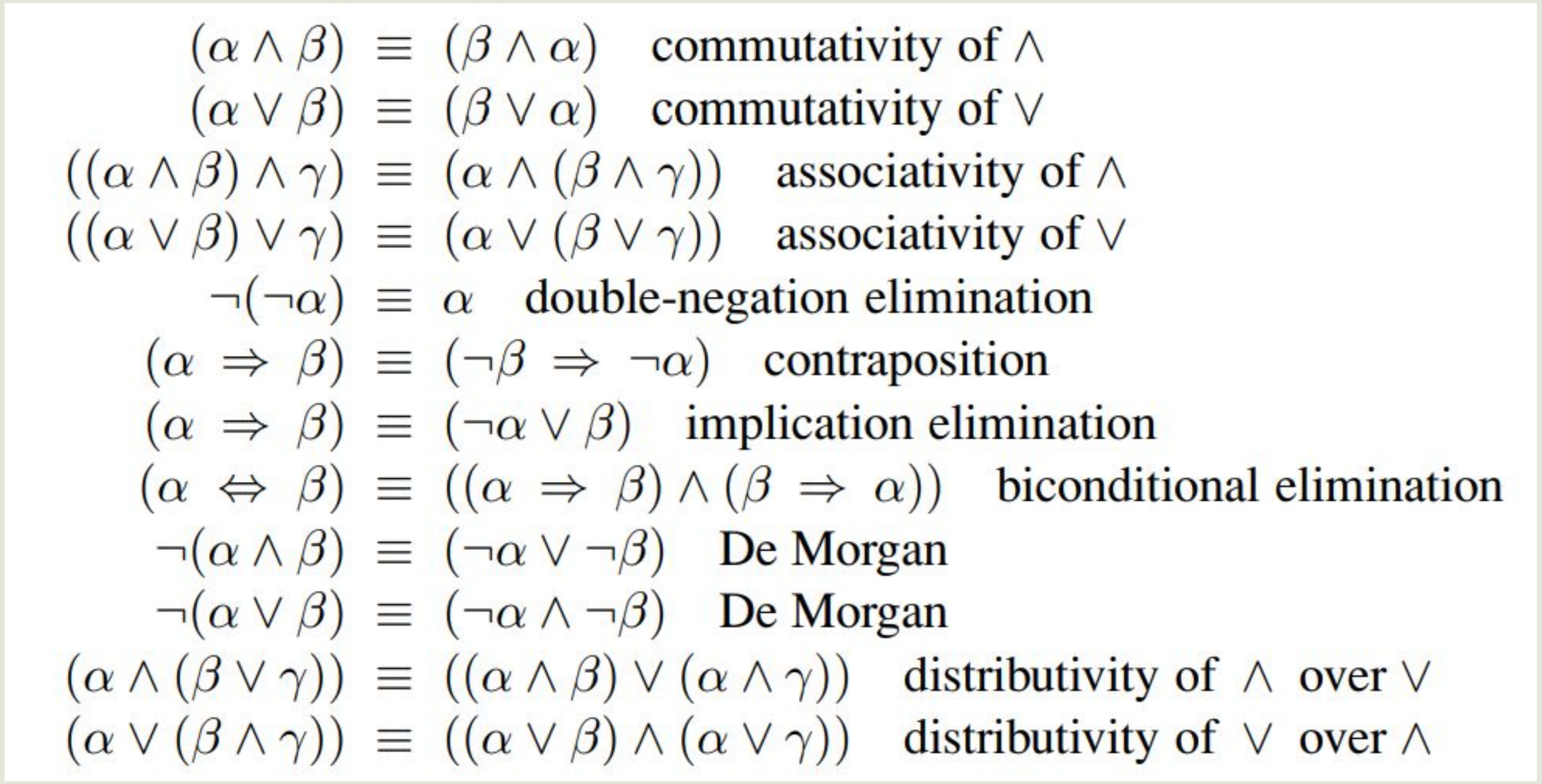


$B_{1,1}$	$B_{2,1}$	$P_{1,1}$	$P_{1,2}$	$P_{2,1}$	$P_{2,2}$	$P_{3,1}$	R_1	R_2	R_3	R_4	R_5	KB
false	*false*	*false*	*false*	*false*	*false*	*false*	*true*	*true*	*true*	*true*	*false*	*false*
false	*false*	*false*	*false*	*false*	*false*	*true*	*true*	*true*	*false*	*true*	*false*	*false*
⋮	⋮	⋮	⋮	⋮	⋮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

Entailment by theorem proving

- In Theorem Proving we apply rules of inference directly to the sentences in our knowledge base to construct a proof of the desired sentence without consulting models. If the number of models is large but the length of the proof is short, then theorem proving can be more efficient than model checking.
- Logical Equivalence: Two sentences α and β are logically equivalent if they are true in the same set of models. We write this as $\alpha \equiv \beta$. For example, $P \land Q$ and $Q \land P$ are logically equivalent;
- Another Definition: $\alpha \equiv \beta$ if and only if $\alpha \models \beta$ and $\beta \models \alpha$.
- A sentence is **valid** if it is true in all models. For example, the sentence $P \lor \neg P$ is valid. Valid sentences are also known as tautologies. they are necessarily true.
- A sentence is **satisfiable** if it is true in, or satisfied by, some model.
  - the knowledge base given earlier, $(R1 \land R2 \land R3 \land R4 \land R5)$, is satisfiable because there are three models in which it is true.

Standard Logical Equivalence



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

Inference rules

- Patterns of inference that can be applied to derive chains of conclusions that lead to the desired goal.
- Modus Ponens
  - Given: α ⇒ β, α, derive β.
  - whenever any sentences of the form α ⇒ β and α are given, then the sentence β can be inferred.
  - if (WumpusAhead ∧ WumpusAlive) ⇒ Shoot and (WumpusAhead ∧ WumpusAlive) are given, then Shoot can be inferred.
- And-Elimination: from a conjunction, any of the conjuncts can be inferred.
  - Given: S1 ∧ S2, derive S1
  - Given: S1 ∧ S2, derive S2
  - (WumpusAhead ∧ WumpusAlive), WumpusAlive can be inferred.
- By considering the possible truth values of α and β, it can be shown that Modus Ponens and And-Elimination are sound. These rules can then be used in any particular instances where they apply, generating sound inferences without the need for enumerating models.
- DeMorgan's Law
  - Given: ¬(A ∧ B) derive ¬ A ∨ ¬ B
  - Given: ¬(A ∨ B) derive ¬ A ∧ ¬ B

Use in Wumpus World

- We start with the knowledge base containing R1 through R5 and show how to prove ¬P1,2, that is, there is no pit in [1,2].
- R1 : $\neg P_{1,1}$.
- R2 : $B_{1,1} \Leftrightarrow (P_{1,2} \lor P_{2,1})$.
- R3 : $B_{2,1} \Leftrightarrow (P_{1,1} \lor P_{2,2} \lor P_{3,1})$.
- R4 : $\neg B_{1,1}$
- R5 : $B_{2,1}$
- First, we apply bi-conditional elimination to R2 to obtain
- R6 : $(B_{1,1} \Rightarrow (P_{1,2} \lor P_{2,1})) \land ((P_{1,2} \lor P_{2,1}) \Rightarrow B_{1,1})$

Use in Wumpus World

- Then we apply And-Elimination to R6 to obtain
  - R7 : $((P_{1,2} \lor P_{2,1}) \Rightarrow B_{1,1})$
- Logical equivalence for contrapositives gives
  - R8 : $(\neg B_{1,1} \Rightarrow \neg(P_{1,2} \lor P_{2,1}))$
- Now we can apply Modus Ponens with R8 and the percept R4 (i.e., $\neg B_{1,1}$), to obtain
  - R9 : $\neg(P_{1,2} \lor P_{2,1})$.
- Finally, we apply De Morgan’s rule, giving the conclusion
  - R10 : ¬P1,2 ∧ ¬P2,1 .
- That is, neither [1,2] nor [2,1] contains a pit.

Well Formed Problem

- to find a sequence of steps that constitutes a proof from searching algorithms:
  - 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 of the inference rule.
  - GOAL: the goal is a state that contains the sentence we are trying to prove.
- Finding a proof can be more efficient because the proof can ignore irrelevant propositions, no matter how many of them there are.
- For example, the proof given earlier leading to $\neg P_{1,2} \land \neg P_{2,1}$ does not mention the propositions $B_{2,1}$, $P_{1,1}$, $P_{2,2}$, or $P_{3,1}$. They can be ignored because the goal proposition, $P_{1,2}$ appears only in sentence R2; the other propositions in R2 appear only in R4 and R2; so R1, R3, and R5 have no bearing on the proof.

monotonicity

- The set of entailed sentences can only increase as information is added to the knowledge base. If we have a proof, adding information to the KB will not invalidate the proof.
- if KB $\models \alpha$ then KB $\wedge\ \beta \models \alpha$.
- For example, suppose the knowledge base contains the additional assertion β stating that there are exactly eight pits in the world. This knowledge might help the agent draw additional conclusions, but it cannot invalidate any conclusion α already inferred—such as the conclusion that there is no pit in [1,2].
- Monotonicity means that inference rules can be applied whenever suitable premises are found in the knowledge base—the conclusion of the rule must follow regardless of what else is in the knowledge base.

Proof by resolution

- A single inference rule, **resolution**, that yields a complete inference algorithm when coupled with any complete search algorithm. a simple version of the resolution rule in the wumpus world.
- We add the following facts to the knowledge base:
- R_{11} : $\neg B_{1,2}$.
- R_{12} : $B_{1,2} \Leftrightarrow (P_{1,1} \lor P_{2,2} \lor P_{1,3})$
- we can now derive the absence of pits in [2,2] and [1,3] (remember that [1,1] is already known to be pit-less):
- R_{13} : $\neg P_{2,2}$
- R_{14} : $\neg P_{1,3}$

Proof by resolution

- We can also apply biconditional elimination to R_3, followed by Modus Ponens with R_5, to obtain the fact that there is a pit in [1,1], [2,2], or [3,1]:
- $R_3 : B_{2,1} \Leftrightarrow (P_{1,1} \lor P_{2,2} \lor P_{3,1})$.
- $R_5 : B_{2,1}$
- $R_{15} : P_{1,1} \lor P_{2,2} \lor P_{3,1}$.
- Now comes the first application of the resolution rule: the literal $\neg P_{2,2}$ in R_{13} resolves with the literal $P_{2,2}$ in R_{15} to give the resolvent
- $R_{16} : P_{1,1} \lor P_{3,1}$
- In English; if there's a pit in one of [1,1], [2,2], and [3,1] and it's not in [2,2], then it's in [1,1] or [3,1]

Proof by resolution

- Similarly, the literal $\neg P_{1,1}$ in R_1 resolves with the literal $P_{1,1}$ in R_{16} to give
- R_{17} : $P_{3,1}$.
- In English: if there's a pit in [1,1] or [3,1] and it's not in [1,1], then it's in [3,1].
- These last two inference steps are examples of the unit resolution inference rule. The unit resolution rule takes a **clause**—a disjunction of literals—and a literal and produces a new clause.
- where each l is a literal l_i and and m are complementary literals

$$\frac{\ell_1 \vee \cdots \vee \ell_k, \qquad m}{\ell_1 \vee \cdots \vee \ell_{i-1} \vee \ell_{i+1} \vee \cdots \vee \ell_k},$$

full resolution 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. This says that resolution takes two clauses and produces a new clause containing all the literals of the two original clauses except the two complementary literals.

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

full resolution rule

- The resulting clause should contain only one copy of each literal. The removal of multiple copies of literals is called factoring.
- For example, if we resolve $(A \lor B)$ with $(A \lor \neg B)$, we obtain $(A \lor A)$, which is reduced to just A.
- The soundness of the resolution rule can be seen easily by considering the literal L_i that is complementary to literal m_j in the other clause.
- If L_i is true, then m_j is false, and hence $m1 \lor \cdots \lor m_{j-1} \lor m_{j+1} \lor \cdots \lor m_n$ must be true, because $m1 \lor \cdots \lor m_n$ is given.
- If L_i is false, then $L_1 \lor \cdots \lor L_{i-1} \lor L_{i+1} \lor \cdots \lor L_k$ must be true because $L_1 \lor \cdots \lor L_k$ is given. Now L_i is either true or false, so one or other of these conclusions holds—exactly as the resolution rule states.
- A resolution-based theorem prover can, for any sentences α and β in propositional logic, decide whether $\alpha \models \beta$

Conjunctive normal form

- The resolution rule applies only to clauses (that is, disjunctions of literals), so it would seem to be relevant only to knowledge bases and queries consisting 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** or **CNF**

Converting a Sentence into CNF

- We illustrate the procedure by converting the sentence $B_{1,1} \Leftrightarrow (P_{1,2} \lor P_{2,1})$ into CNF. The steps are as follows:
- 1. Eliminate $\Leftrightarrow$, replacing $\alpha \Leftrightarrow \beta$ with $(\alpha \Rightarrow \beta) \land (\beta \Rightarrow \alpha)$.
    - $(B_{1,1} \Rightarrow (P_{1,2} \lor P_{2,1})) \land ((P_{1,2} \lor P_{2,1}) \Rightarrow B_{1,1})$.
- 2. Eliminate $\Rightarrow$, replacing $\alpha \Rightarrow \beta$ with $\neg\alpha \lor \beta$:
    - $(\neg B_{1,1} \lor P_{1,2} \lor P_{2,1}) \land (\neg(P_{1,2} \lor P_{2,1}) \lor B_{1,1})$
- 3. CNF requires ¬ to appear only in literals, so we “move ¬ inwards”
    - $\neg(\neg\alpha) \equiv \alpha$ (double-negation 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)
    - In the example, we require just one application of the last rule:
    - $(\neg B_{1,1} \lor P_{1,2} \lor P_{2,1}) \land ((\neg P_{1,2} \land \neg P_{2,1}) \lor B_{1,1})$.

Converting a Sentence into CNF

- 4. Now we have a sentence containing nested $\wedge$ and $\vee$ operators applied to literals. We apply the distributive law, distributing $\vee$ over $\wedge$ wherever possible.
- $(\neg B_{1,1} \vee P_{1,2} \vee P_{2,1}) \wedge (\neg P_{1,2} \vee B_{1,1}) \wedge (\neg P_{2,1} \vee B_{1,1})$.
- The original sentence is now in CNF, as a conjunction of three clauses.

Grammar of CNF

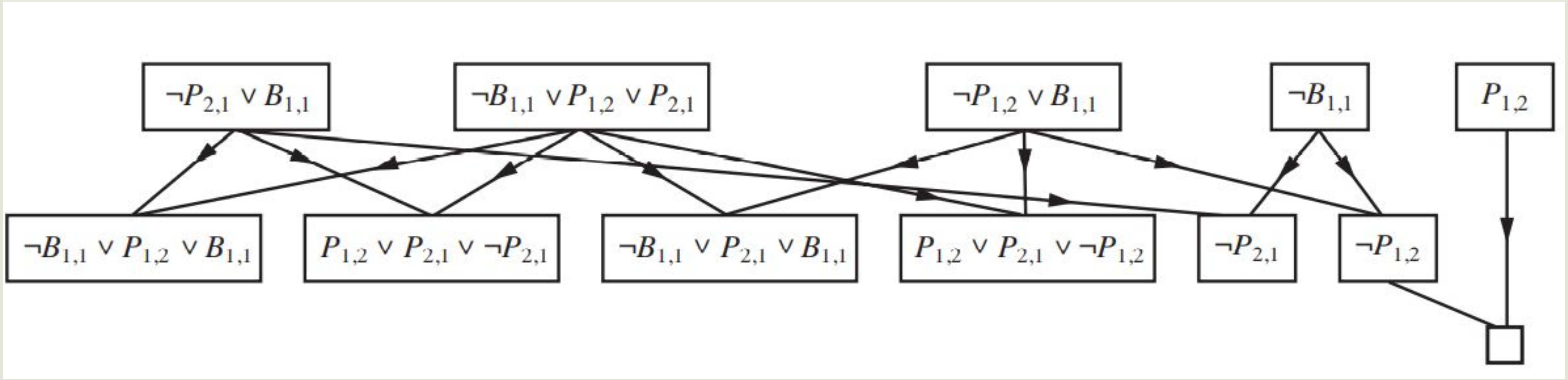
$$
\begin{array}{rcl}
\mathit{CNFSentence} & \rightarrow & \mathit{Clause}_1 \wedge \cdots \wedge \mathit{Clause}_n \\
\mathit{Clause} & \rightarrow & \mathit{Literal}_1 \vee \cdots \vee \mathit{Literal}_m \\
\mathit{Literal} & \rightarrow & \mathit{Symbol} \mid \neg \mathit{Symbol} \\
\mathit{Symbol} & \rightarrow & P \mid Q \mid R \mid \ldots \\
\mathit{HornClauseForm} & \rightarrow & \mathit{DefiniteClauseForm} \mid \mathit{GoalClauseForm} \\
\mathit{DefiniteClauseForm} & \rightarrow & (\mathit{Symbol}_1 \wedge \cdots \wedge \mathit{Symbol}_l) \Rightarrow \mathit{Symbol} \\
\mathit{GoalClauseForm} & \rightarrow & (\mathit{Symbol}_1 \wedge \cdots \wedge \mathit{Symbol}_l) \Rightarrow \mathit{False}
\end{array}
$$

A resolution algorithm

- Inference procedures based on resolution work by using the principle of proof by contradiction. That is, to show that KB $\models \alpha$, we show that (KB $\land \neg\alpha$) is unsatisfiable.
- First, (KB $\land \neg\alpha$) is converted into CNF. Then, the resolution rule is applied to the resulting clauses.
- Each pair that contains complementary literals is resolved to produce a new clause, which is added to the set if it is not already present. The process continues until one of two things happens:
  - there are no new clauses that can be added, in which case KB does not entail α;
  - two clauses resolve to yield the empty clause, in which case KB entails α.
- An empty clause represents a contradiction is to observe that it arises only from resolving two complementary unit clauses such as P and ¬P.

Application of resolution algorithm

- When the agent is in [1,1], there is no breeze, so there can be no pits in neighboring squares. The relevant knowledge base is
- KB = $R_2 \wedge R_4 = (B_{1,1} \Leftrightarrow (P_{1,2} \vee P_{2,1})) \wedge \neg B_{1,1.}$
- and we wish to prove α which is, say, ¬P1,2.


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

Horn clauses and definite clauses

- One restricted form resolution is the definite clause, which is a disjunction of literals of which **exactly one is positive**. For example, the clause $(\neg L_{1,1} \lor \neg Breeze \lor B_{1,1})$ is a definite clause, whereas $(\neg B_{1,1} \lor P_{1,2} \lor P_{2,1})$ is not.
- Horn clause, which is a disjunction of literals of which at most one is positive. So, all definite clauses are Horn clauses,
- The clauses with no positive literals; these are called goal clauses.
- Horn clauses are closed under resolution: if you resolve two Horn clauses, you get back a Horn clause.

Use of Definite clauses

- 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.
- For example, the definite clause $(\neg L_{1,1} \vee \neg Breeze \vee B_{1,1})$ can be written as the implication
- $(L_{1,1} \wedge Breeze) \Rightarrow B_{1,1.}$ it says that if the agent is in [1,1] and there is a breeze, then [1,1] is breezy.
- In Horn form, the premise is called the **body** and the conclusion is called the **head**. A sentence consisting of a single positive literal, such as $L_{1,1}$, is called a fact. It too can be written in implication form as $True \Rightarrow L_{1,1}$, but it is simpler to write just $L_{1,1.}$
- Inference with Horn clauses can be done through the forward-chaining and backward chaining algorithms.
- Deciding entailment with Horn clauses can be done in time that is linear in the size of the knowledge base

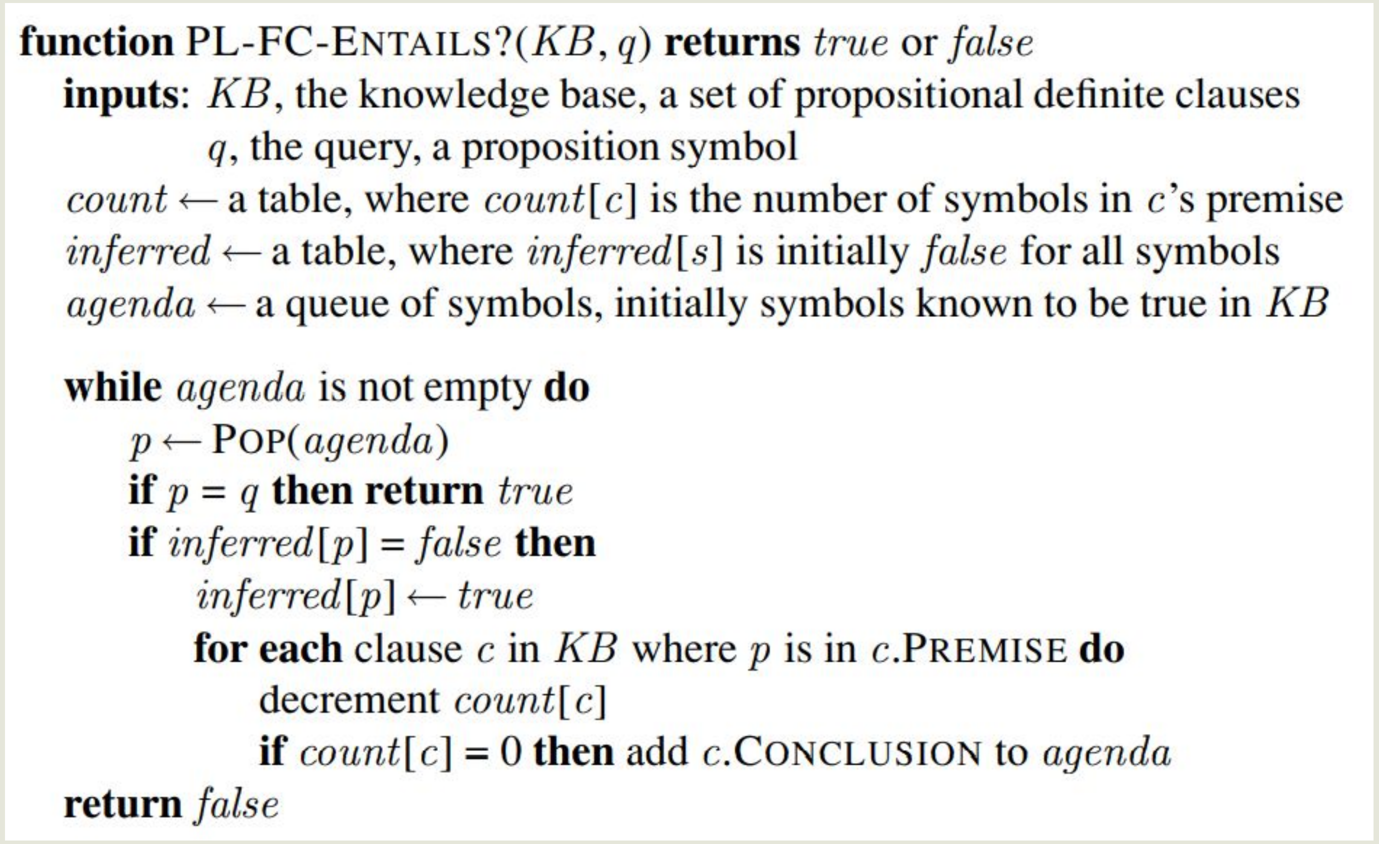
Forward and backward chaining

- The forward-chaining algorithm determines if a single proposition symbol q—the query—is entailed by a knowledge base of definite clauses.
- It begins from known facts (positive literals) in the knowledge base. If all the premises of an implication are known, then its conclusion is added to the set of known facts.
- For example, if $L_{1,1}$ and Breeze are known and $(L_{1,1} \land \text{Breeze}) \Rightarrow B_{1,1}$ is in the knowledge base, then $B_{1,1}$ can be added.
- This process continues until the query q is added or until no further inferences can be made.

Forward Chaining

- The forward-chaining algorithm for propositional logic.
- The agenda keeps track of symbols known to be true but not yet “processed.”
- The count table keeps track of how many premises of each implication are as yet unknown. Whenever a new symbol p from the agenda is processed, the count is reduced by one for each implication in whose premise p appears.
- If a count reaches zero, all the premises of the implication are known, so its conclusion can be added to the agenda.
- Finally, we need to keep track of which symbols have been processed; a symbol that is already in the set of inferred symbols need not be added to the agenda again. This avoids redundant work and prevents loops caused by implications such as $P \Rightarrow Q$ and $Q \Rightarrow P$.

Forward Chaining

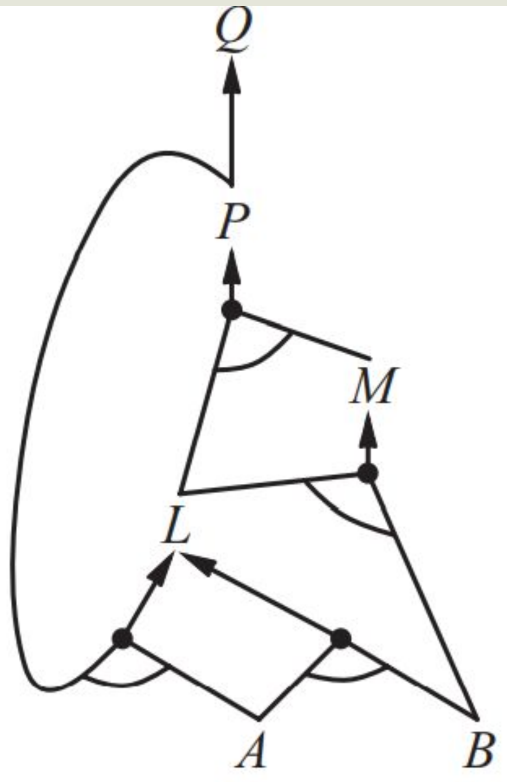


```
function PL-FC-ENTAILS?($KB$, $q$) returns $true$ or $false$
  inputs: $KB$, the knowledge base, a set of propositional definite clauses
          $q$, the query, a proposition symbol
  $count$ ← a table, where $count[c]$ is the number of symbols in $c$'s premise
  $inferred$ ← a table, where $inferred[s]$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

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

(a)



(b)