Chapter 7 - Logical Agents

Russell, S., & Norvig, P. (2022). *Artificial Intelligence - A Modern Approach* (4th global ed.). Pearson.

Contents

- 1. Knowledge-Based Agents
- 2. The Wumpus World
- 3. Logic
- 4. Propositional Logic
- 5. Propositional Theorem Proving

Logical Agents

- In which we design agents that can form representations of a complex world, use a process of inference to derive new representations about the world, and use these new representations to deduce what to do.
- Humans, it seems, know things; and what they know helps them do things. In AI, **knowledge-based agents** use a process of **reasoning** over an internal **representation** of knowledge to decide what actions to take.

1. Knowledge-Based Agents

- The central component of a knowledge-based agent is its **knowledge base**, or *KB*.
- 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 being given without being derived from other sentences, we call it an **axiom**.
- **TELL** operation: Add a new sentence to the knowledge base.
- **ASK** operation: Query what is known.

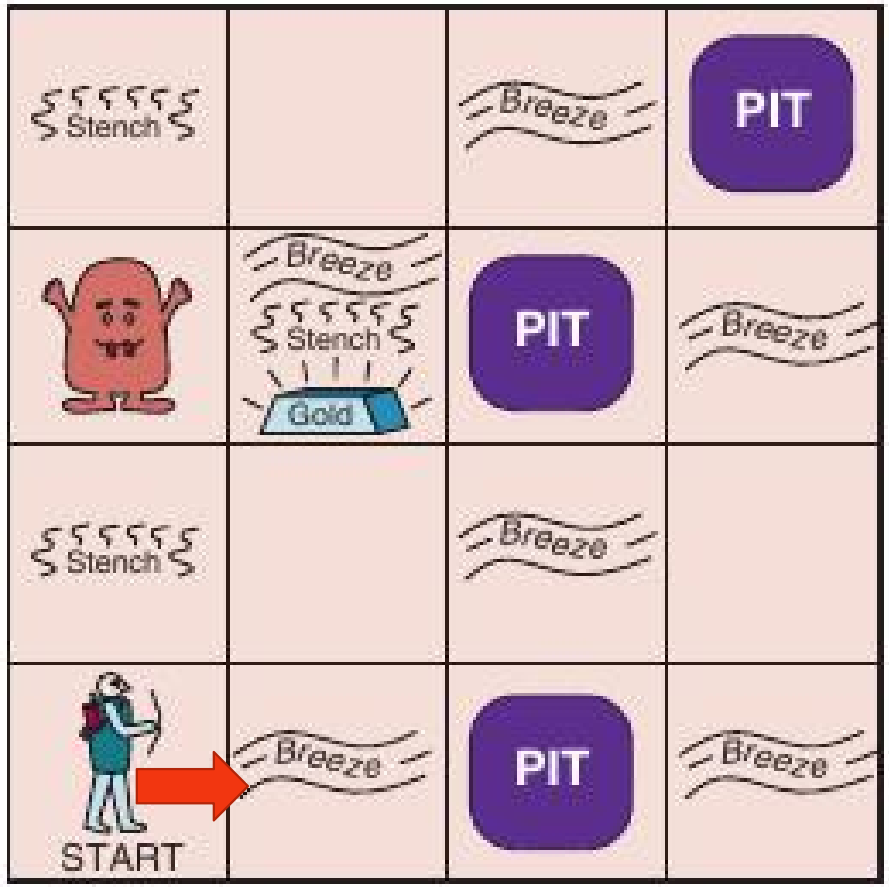
1. Knowledge-Based Agents

```
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
```

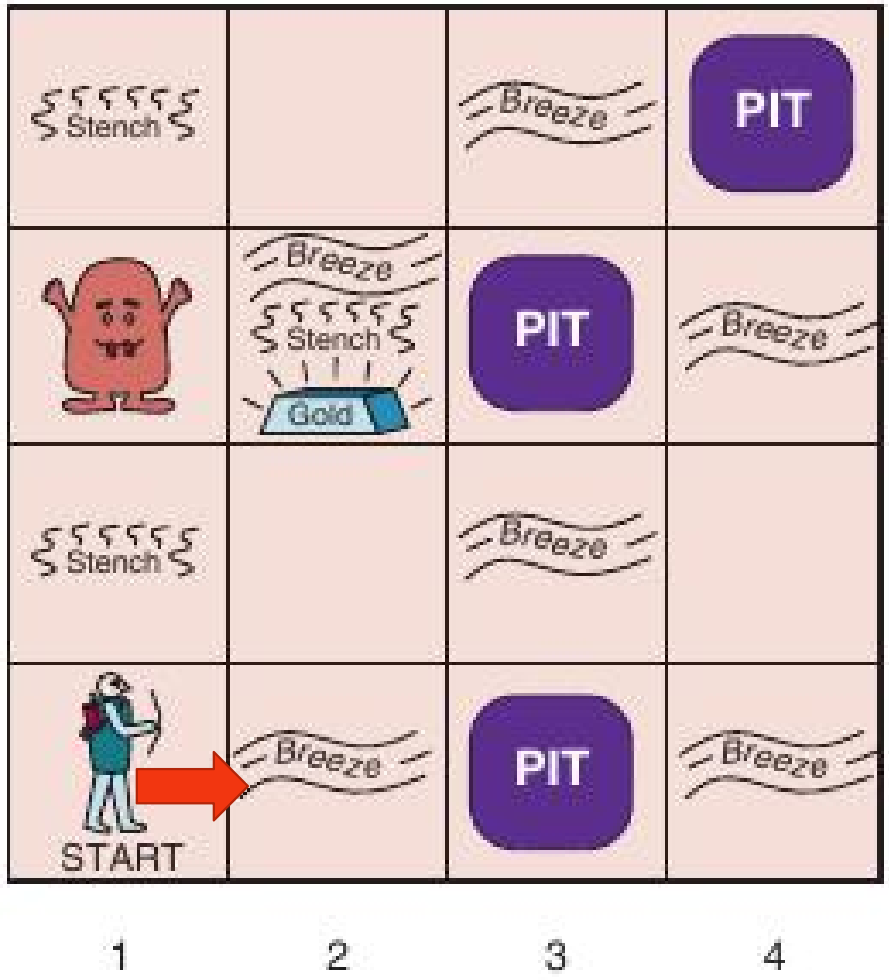
- **Figure 7.1** A generic knowledge-based agent.
- Given a percept, the agent adds the percept to its knowledge base, asks the knowledge base for the best action, and tells the knowledge base that it has in fact taken that action.

2. The Wumpus World



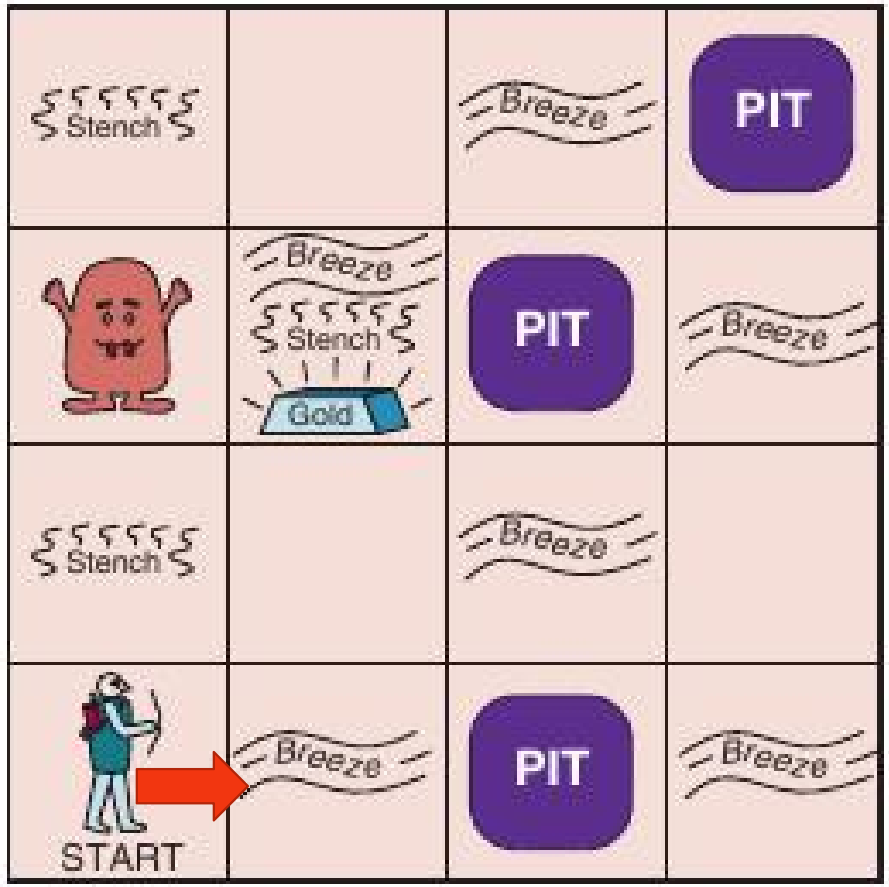
- **Figure 7.2** A typical wumpus world. The agent is in the bottom left corner, facing east (rightward).
- The task environment is given by the PEAS description:
  - **Performance measure**: +1000 for climbing out of the cave with the gold, –1000 for falling into a pit or being eaten by the wumpus, –1 for each action taken, and –10 for using up the arrow.
    - The game ends either when the agent dies or when the agent climbs out of the cave.

2. The Wumpus World



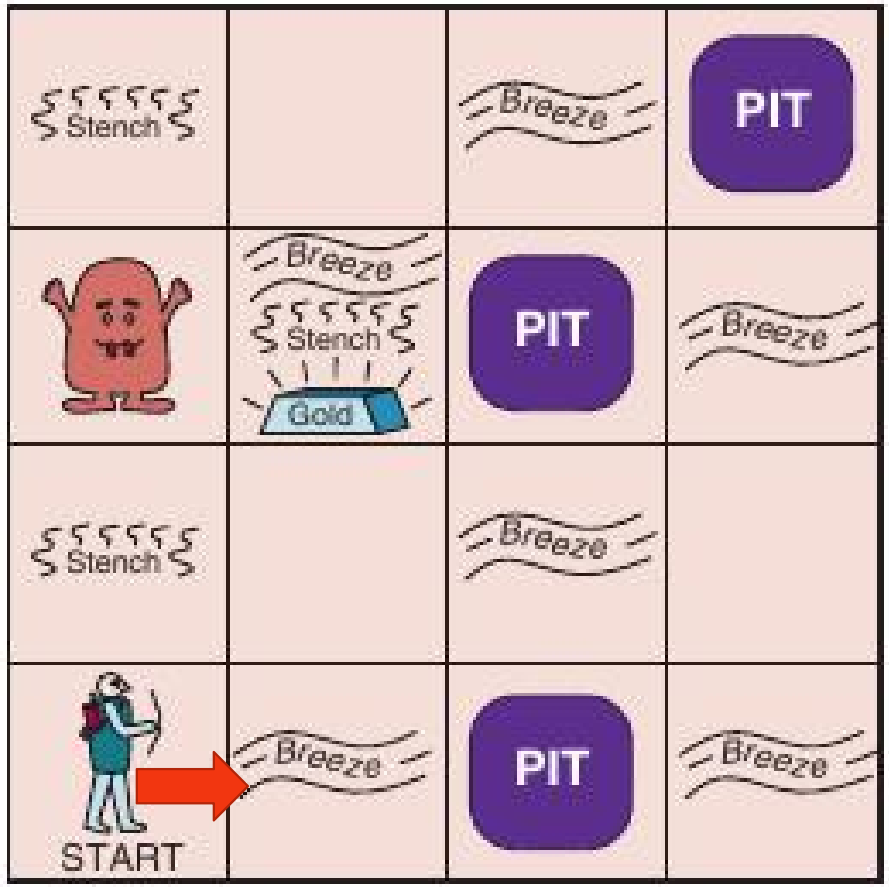
- **Environment**: A 4 × 4 grid of rooms, with walls surrounding the grid.
  - The agent always starts in the square labeled [1,1], facing to the east.
  - The locations of the gold and the wumpus are chosen randomly, with a uniform distribution, from the squares other than the start square.
  - In addition, each square other than the start can be a pit, with probability 0.2.

2. The Wumpus World



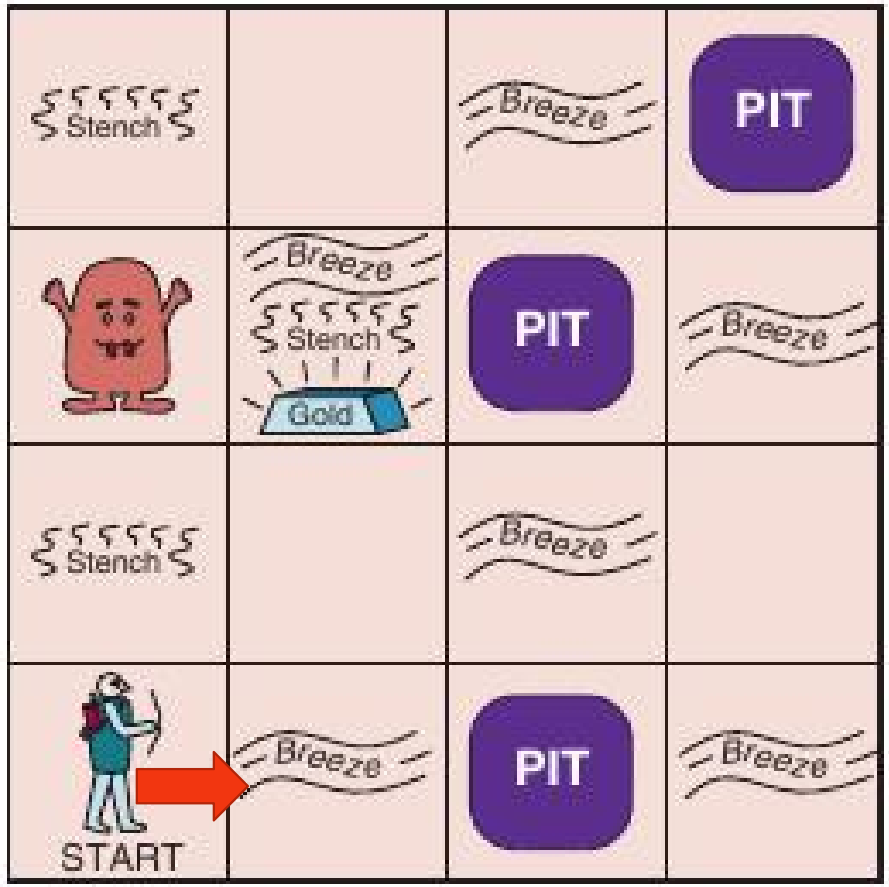
- **Actuators**: The agent can move *Forward*, *TurnLeft* by 90◦ , or *TurnRight* by 90◦. The agent dies a miserable death if it enters a square containing a pit or a live wumpus. (It is safe, albeit smelly, to enter a square with a dead wumpus.)
  - If an agent tries to move forward and bumps into a 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 action *Shoot* can be used to fire an arrow in a straight line in the direction the agent is facing. The arrow continues until it either hits (and hence kills) the wumpus or hits a wall. 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].

2. The Wumpus World



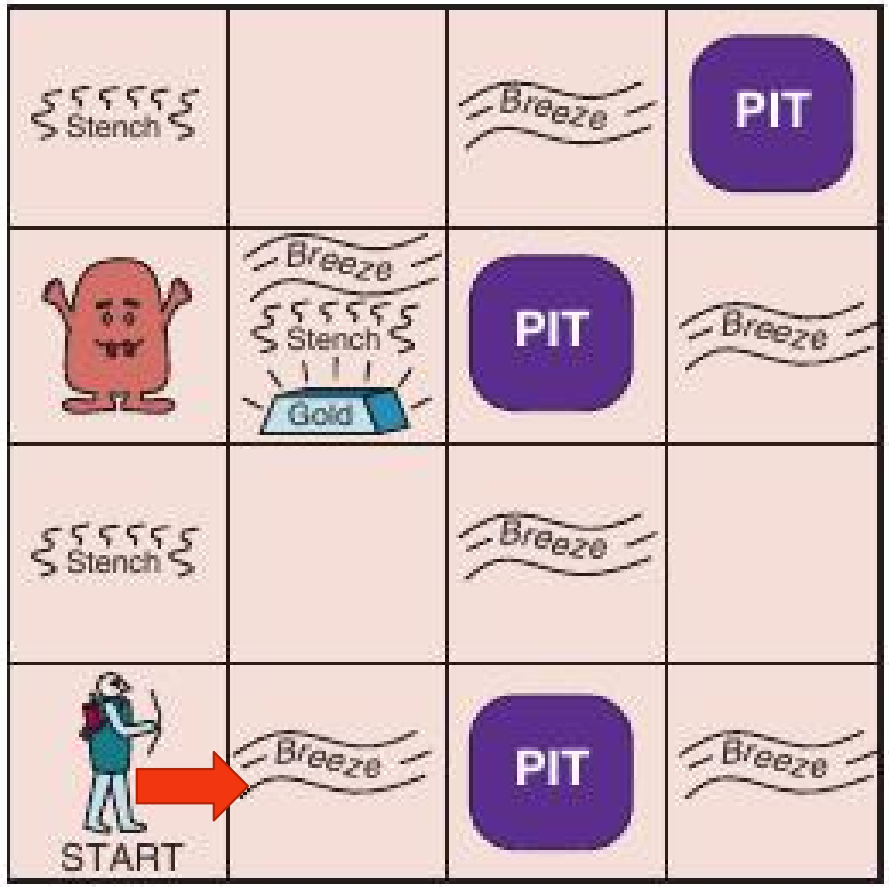
- **Sensors**: The agent has five sensors, each of which gives a single bit of information:
  - In the squares directly (not diagonally) adjacent to the wumpus, the agent will perceive a *Stench*.
  - In the squares directly adjacent to a pit, the agent will perceive a *Breeze*.
  - In the square where 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.
  - 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*].

2. The Wumpus World



- The wumpus environment
  - Deterministic, Discrete, Static, Single-agent (the wumpus doesn't move)
  - Sequential (because rewards may come only after any actions are taken)
  - Partially observable (some aspects of the state are not directly perceivable: the agent's location, the wumpus's state of health, and the available of an arrow)

2. The Wumpus World



- The wumpus environment
  - As for the locations of the pits and the wumpus:
    - We could treat them as unobserved parts of the state—in which case, the transition model for the environment is completely known, and finding the locations of pits completes the agent's knowledge of the state.
    - Alternatively, we could say that the transition model itself is unknown because the agent doesn't know which *Forward* actions are fatal—in which case, discovering the locations of pits and wumpus completes the agent's knowledge of the transition model.

2. The Wumpus World

- For an agent in the environment, the main challenge is its *initial ignorance* of the configuration of the environment; overcoming this ignorance seems to require logical reasoning.
- In most instances of the wumpus world, it is possible for the agent to retrieve the gold safely.
- Occasionally, the agent must choose between going home empty-handed and risking death to find the gold.
  - About 21% of the environments are utterly unfair, because the gold is in a pit or surrounded by pits.

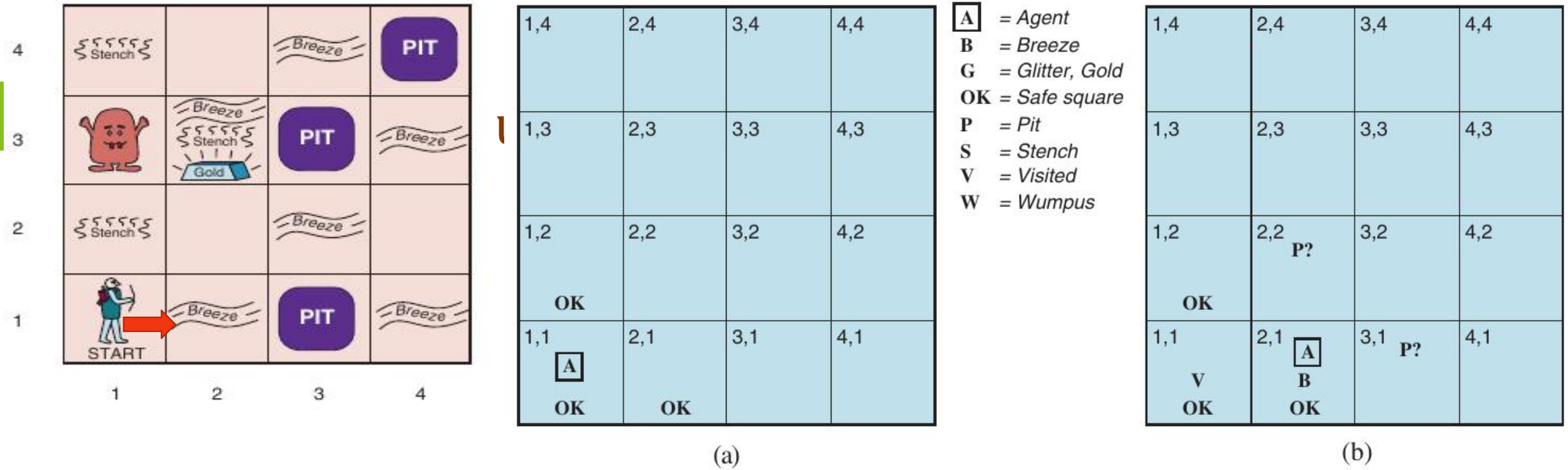
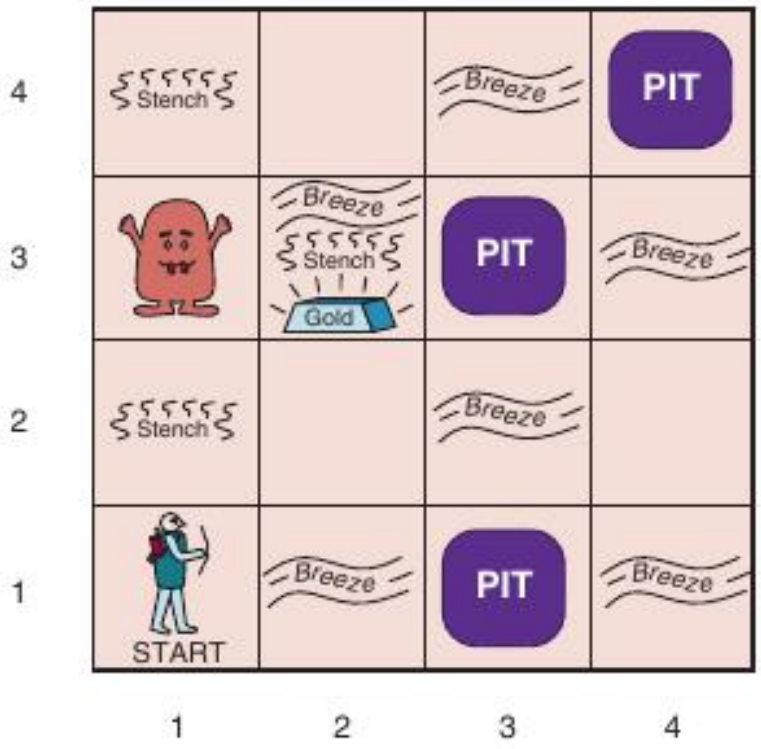


Figure 7.3 The first step taken by the agent in the wumpus world. (a) The initial situation, after percept [*None*, *None*, *None*, *None*, *None*]. (b) After moving to [2,1] and perceiving [*None*, *Breeze*, *None*, *None*, *None*].

- Let us watch a knowledge-based wumpus agent *exploring the environment*. We use an informal knowledge representation language consisting of writing down symbols in a grid.
- The agent's initial knowledge base contains the rules of the environment, as described previously; in particular, it knows that it is in [1,1] and that [1,1] is a safe square; we denote that with an "A" and "OK," respectively, in square [1,1].



1,4	2,4	3,4	4,4
1,3 W!	2,3	3,3	4,3
1,2 A S OK	2,2 OK	3,2	4,2
1,1 V OK	2,1 B V OK	3,1 P!	4,1

(a)

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

1,4	2,4 P?	3,4	4,4
1,3 W!	2,3 A S G B	3,3 P?	4,3
1,2 S V OK	2,2 V OK	3,2	4,2
1,1 V OK	2,1 B V OK	3,1 P!	4,1

(b)

Figure 7.4 Two later stages in the progress of the agent. (a) After moving to [1,1] and then [1,2], and perceiving [*Stench*, *None*, *None*, *None*, *None*]. (b) After moving to [2,2] and then [2,3], and perceiving [*Stench*, *Breeze*, *Glitter*, *None*, *None*].

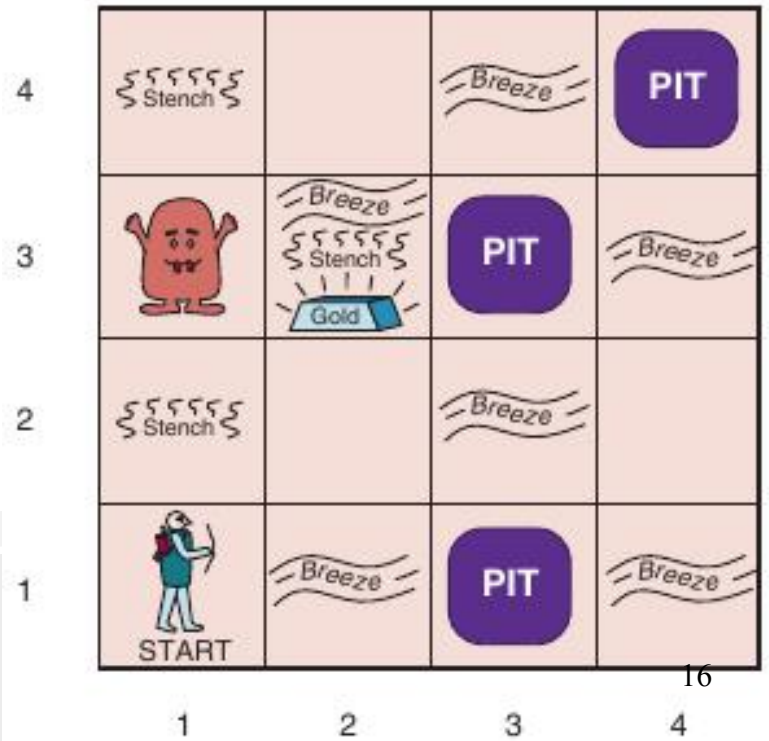
- Note that in each case for which the agent draws a conclusion from the available information, that conclusion is *guaranteed* to be correct if the available information is correct. This is a fundamental property of **logical reasoning**.

3. Logic

- 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.
  - For example, the semantics for arithmetic specifies that the sentence "$x + y = 4$" is true in a world where x is 2 and y is 2, but false in a world where x is 1 and y is 1.
- In standard logics, every sentence must be either true or false in each possible world—there is no "in between."
- We use the term **model** in place of "possible world." Whereas possible worlds might be thought of as (potentially) real environments that the agent might or might not be in, models are mathematical abstractions, each of which has a fixed truth value (true or false) for every relevant sentence.
- Formally, the possible models are just all possible assignments of nonnegative integers to the variables x and y. Each such assignment determines 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.

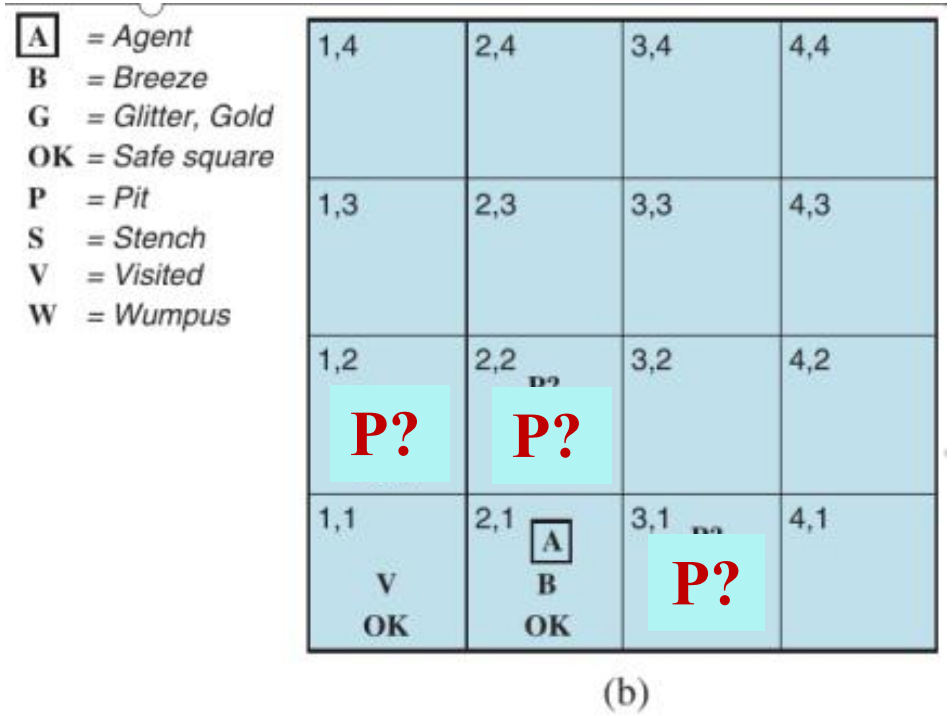
3. Logic

- **Logical entailment**: $\alpha \models \beta$: 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 $M(\alpha) \subseteq M(\beta)$.
- The KB can be thought of as a 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.
  - For example, the KB is false in any model in which [1,2] contains a pit, because there is no breeze in [1,1].

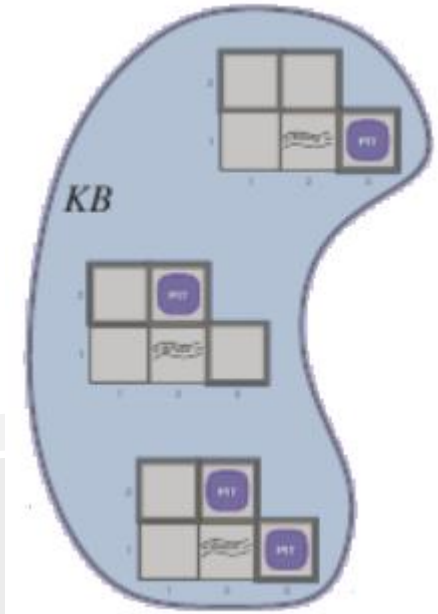


3. Logic

- Consider the situation: the agent has detected nothing in [1,1] and a breeze in [2,1]
  - 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, so (ignoring other aspects of the world for now) there are $2^3 = 8$ possible models.
  - There are in fact just three models in which the *KB* is true, and these are shown surrounded by a solid line in Figure 7.5.



(b)



3. Logic

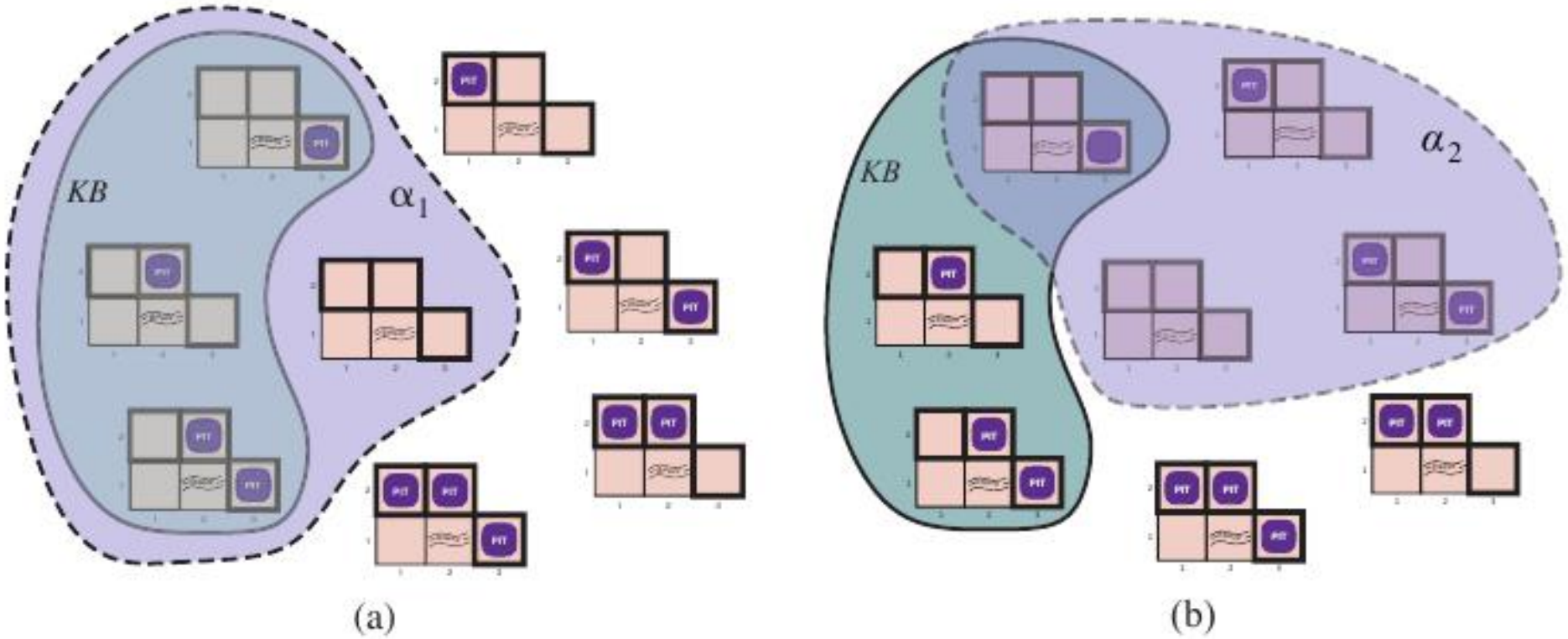


Figure 7.5 Possible models for the presence of pits in squares [1,2], [2,2], and [3,1]. The KB corresponding to the observations of nothing in [1,1] and a breeze in [2,1] is shown by the solid line. (a) Dotted line shows models of α_1 (no pit in [1,2]). (b) Dotted line shows models of α_2 (no pit in [2,2]).

Logical inference

- Now let us consider the possible conclusion: α_1 = "There is no pit in [1,2]."
  - Figure 7.5(a): In every model in which KB is true, α_1 is also true.
  - Hence, $KB \models \alpha_1$: there is no pit in [1,2].

3. Logic

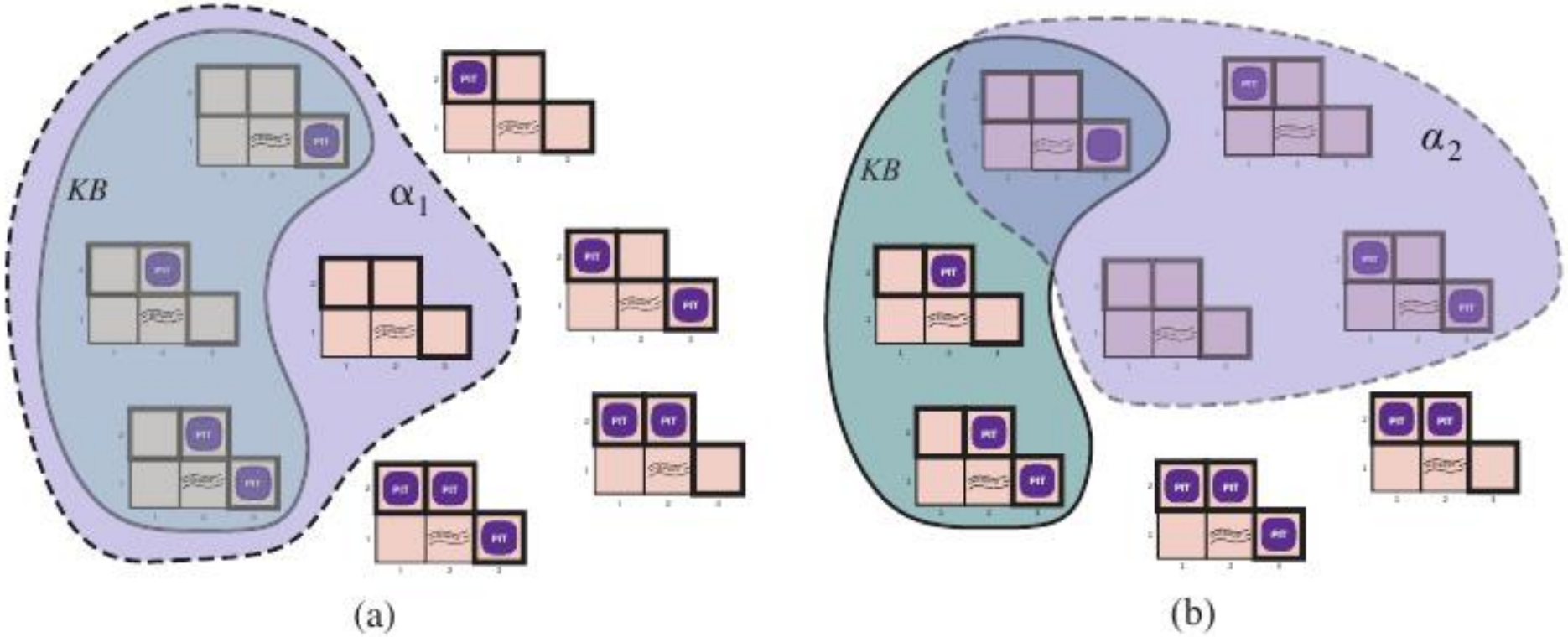


Figure 7.5 Possible models for the presence of pits in squares [1,2], [2,2], and [3,1]. The KB corresponding to the observations of nothing in [1,1] and a breeze in [2,1] is shown by the solid line. (a) Dotted line shows models of α_1 (no pit in [1,2]). (b) Dotted line shows models of α_2 (no pit in [2,2]).

Logical inference

- Now let us consider the possible conclusion: α_2 = "There is no pit in [2,2]."
  - Figure 7.5(b): In some models in which KB is true, α_2 is false.
  - Hence, $KB \not\models \alpha_2$: Hence, KB does not entail α_2. The agent *cannot* conclude that there is no pit in [2,2]. (Nor can it conclude that there is a pit in [2,2].)

3. Logic

- The inference algorithm illustrated in Figure 7.5 is called **model checking**, because it enumerates all possible models to check that α is true in all models in which KB is true, that is, that $M(KB) \subseteq M(\alpha)$.
- If an inference algorithm i can derive α from KB:
  - $KB \vdash_i \alpha$
  - α is derived from KB by i.
  - i derives α from KB.

3. Logic

- **Soundness**
  - An inference algorithm that derives only entailed sentences is called **sound** or **truth-preserving**.
  - Soundness is a highly desirable property.
  - An unsound inference procedure essentially makes things up as it goes along—it announces the discovery of nonexistent needles.
  - It is easy to see that model checking, when it is applicable, is a sound procedure.

3. Logic

- **Completeness**
  - An inference algorithm is complete if it can derive any sentence that is entailed.
  - Fortunately, there are complete inference procedures for logics that are sufficiently expressive to handle many knowledge bases.

3. Logic

- We have described a reasoning process whose conclusions are guaranteed to be true in any world in which the premises are true.
  - *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.*

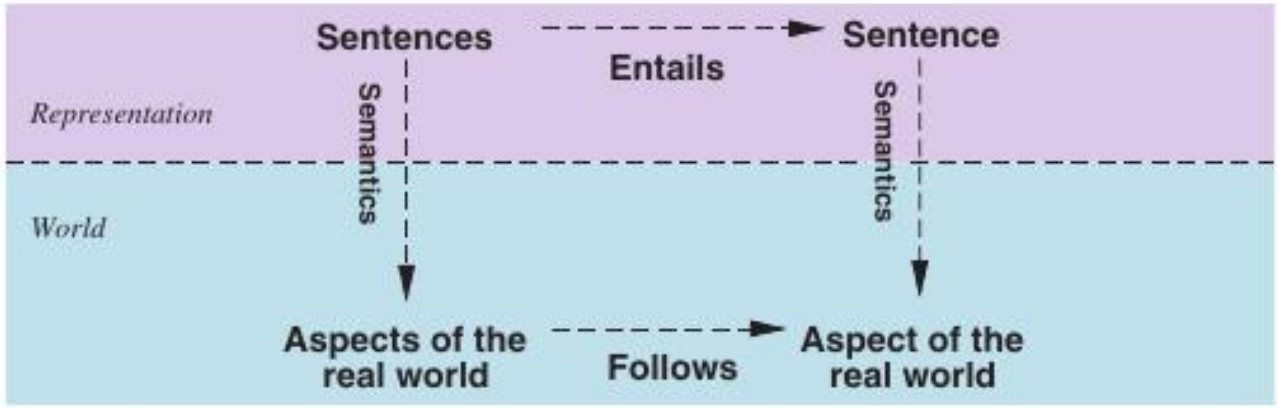


Figure 7.6 Sentences are physical configurations of the agent, and reasoning is a process of constructing new physical configurations from old ones. Logical reasoning should ensure that the new configurations represent aspects of the world that actually follow from the aspects that the old configurations represent.

3. Logic

- **Grounding**—the connection between logical reasoning processes and the real environment in which the agent exists.
  - *How do we know that KB is true in the real world?* (After all, KB is just "syntax" inside the agent's head.)
  - A simple answer is that the agent's sensors create the connection. For example, our wumpus-world agent has a smell sensor.
  - The agent program creates a suitable sentence whenever there is a smell. Then, whenever that sentence is in the knowledge base, it is true in the real world.

4. Propositional Logic: A Very Simple Logic

- Propositional logic
  - Syntax: The structure of sentences
  - Semantics: The way in which the truth of sentences is determined
  - A syntatic algorithm for logical inference that implements the semantic notion of entailment.

4. Propositional Logic: A Very Simple Logic

Syntax

- The syntax of propositional logic defines the allowable sentences.
- **Proposition symbol**
  - Each symbol stands for a proposition that can be true or false.
  - Start with an uppercase letter and may contain other letters or subscripts: *P, Q,* R_1, *FacingEast*
  - There are two proposition symbols with fixed meanings: *True* is the always-true proposition and *False* is the always-false proposition.
- **Atomic sentences** consist of a single **proposition symbol**.

4. Propositional Logic: A Very Simple Logic

Syntax

- **Complex sentences** are constructed from simpler sentences, using parentheses and operators called **logical connectives**.
- There are five connectives in common use:
  - ¬ (not)
  - ∧ (and)
  - ∨ (or)
  - ⇒ (implies)
  - ⇔ (if and only if)

4. Propositional Logic: A Very Simple Logic

Syntax

$$
\begin{aligned}
\textit{Sentence} &\rightarrow \textit{AtomicSentence} \mid \textit{ComplexSentence} \\
\textit{AtomicSentence} &\rightarrow \textit{True} \mid \textit{False} \mid P \mid Q \mid R \mid \ldots \\
\textit{ComplexSentence} &\rightarrow (\textit{Sentence}) \\
&\mid \neg \textit{Sentence} \\
&\mid \textit{Sentence} \wedge \textit{Sentence} \\
&\mid \textit{Sentence} \vee \textit{Sentence} \\
&\mid \textit{Sentence} \Rightarrow \textit{Sentence} \\
&\mid \textit{Sentence} \Leftrightarrow \textit{Sentence} \\
\text{OPERATOR PRECEDENCE} &: \neg, \wedge, \vee, \Rightarrow, \Leftrightarrow
\end{aligned}
$$

Figure 7.7 A BNF (Backus–Naur Form) grammar of sentences in propositional logic, along with operator precedences, from highest to lowest.

4. Propositional Logic: A Very Simple 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 sets the **truth value**—*true* or *false*—for every proposition symbol.
  - For example, 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

$$m_1 = \{P_{1,2} = false, P_{2,2} = false, P_{3,1} = true\}$$

    - With three proposition symbols, there are $2^3 = 8$ possible models.
    - However, the models are purely mathematical objects with no necessary connection to worlds. $P_{1,2}$ is just a symbol; it might mean "there is a pit in [1,2]" or "I'm in Paris today and tomorrow."

4. Propositional Logic: A Very Simple Logic

Semantics

- The semantics for propositional logic must specify how to compute the truth value of *any sentence,* given a model. This is done recursively.
- All sentences are constructed from atomic sentences and the five connectives; therefore, 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.
- Atomic sentences are easy:
  - *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. For example, in the model $m_1 = \{P_{1,2} = false, P_{2,2} = false, P_{3,1} = true\}$, $P_{1,2}$ is false.

4. Propositional Logic: A Very Simple Logic

Semantics

- For complex sentences, we have five rules, which hold for any subsentences P and Q (atomic or complex) 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.

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*

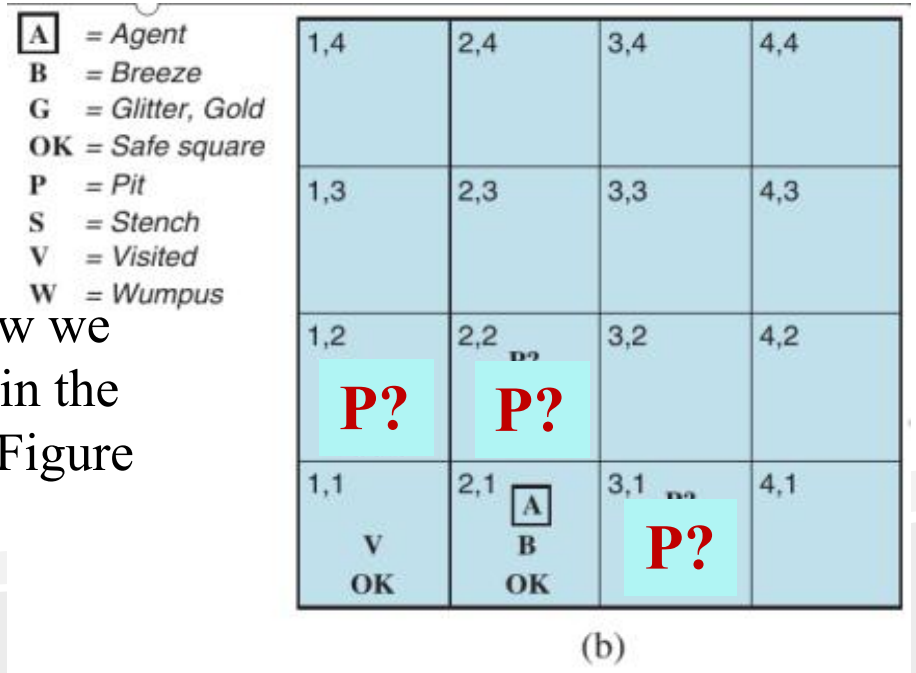
4. Propositional Logic: A Very Simple Logic

A Simple Knowledge Base

- Knowledge base for the wumpus world
- Symbols for each $[x, y]$ location:
  - $P_{x,y}$ 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 there is a breeze in $[x, y]$.
  - $S_{x,y}$ is true if there is a stench in $[x, y]$.
  - $L_{x,y}$ is true if the agent is in location $[x, y]$.

4. Propositional Logic: A Very Simple Logic
A Simple Knowledge Base

- There is no pit in [1,1]:
  - R_1: $\neg P_{1,1}$.
- 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:
  - R_2: $B_{1,1} \Leftrightarrow (P_{1,2} \vee P_{2,1})$.
  - R_3: $B_{2,1} \Leftrightarrow (P_{1,1} \vee P_{2,2} \vee P_{3,1})$.
- The preceding sentences are true in all wumpus worlds. Now we include the breeze percepts for the first two squares visited in the specific world the agent is in, leading up to the situation in Figure 7.3(b).
  - R_4: $\neg B_{1,1}$.
  - R_5: $B_{2,1}$.



(b)

4. Propositional Logic: A Very Simple Logic
A Simple Inference Procedure

- 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?
- Our first algorithm for inference is a model-checking approach 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.
- In the 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 $P_{3,1}$.
  - With seven symbols, there are $2^7 = 128$ possible models in three of these, KB is true (Figure 7.9).
  - In those three models, $\neg P_{1,2}$ is true, hence there is no pit in [1,2]. On the other hand, $P_{2,2}$ is true in two of the three models and false in one, so we cannot yet tell whether there is a pit in [2,2].

4. Propositional Logic: A Very Simple Logic

A Simple Inference Procedure

$B_{1,1}$	$B_{2,1}$	$P_{1,1}$	$P_{1,2}$	$P_{2,1}$	$P_{2,2}$	$P_{3,1}$	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*

Figure 7.9 A truth table constructed for the knowledge base given in the text. *KB* is true if R_1 through R_5 are true, which occurs in just 3 of the 128 rows (the ones underlined in the right-hand column). 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].

4. Propositional Logic: A Very Simple Logic

A Simple Inference Procedure

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

  symbols ← a list of the proposition symbols in KB and α
  return TT-CHECK-ALL(KB, α, symbols, { })

function TT-CHECK-ALL(KB, α, symbols, model) returns true or false
  if EMPTY?(symbols) then
      if PL-TRUE?(KB, model) then return PL-TRUE?(α, model)
      else return true        // when KB is false, always return true
  else
      P ← FIRST(symbols)
      rest ← REST(symbols)
      return (TT-CHECK-ALL(KB, α, rest, model ∪ {P = true})
              and
              TT-CHECK-ALL(KB, α, rest, model ∪ {P = false }))
```

Figure 7.10 A truth-table enumeration algorithm for deciding propositional entailment. (TT stands for truth table.) PL-TRUE? returns *true* if a sentence holds within a model. The variable *model* represents a partial model—an assignment to some of the symbols. The keyword **and** here is an infix function symbol in the pseudocode programming language, not an operator in propositional logic; it takes two arguments and returns *true* or *false*.

4. Propositional Logic: A Very Simple Logic

A Simple Inference Procedure

- 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.
- If KB and α contain n symbol in all, then there are 2^n models. Thus, the time complexity of the algorithm is $O(2^n)$.

5. Propositional Theorem Proving

- Entailment can be done by:
  - **Model checking**: Enumerating models and showing that the sentence must hold in all models.
  - **Theorem proving**: Applying 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.

5. Propositional Theorem Proving

- Additional concepts related to entailment
  - **Logical equivalence**: $\alpha \equiv \beta$
    - Two sentences α and β are logically equivalent if they are true in the same set of models.
    - An alternative definition of equivalence is as follows: any two sentences α and β are equivalent if and only if each of them entails the other:

$$\alpha \equiv \beta \text{ if and only if } \alpha \models \beta \text{ and } \beta \models \alpha.$$

5. Propositional Theorem Proving

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

Figure 7.11 Standard logical equivalences. The symbols α, β, and γ stand for arbitrary sentences of propositional logic.

5. Propositional Theorem Proving

- **Validity**: A sentence is valid if it is true in *all* models.
  - For example, the sentence $P \vee \neg P$ is valid.
  - Valid sentences are also known as **tautologies**—they are *necessarily* true.
  - Because the sentence *True* is true in all models, every valid sentence is logically equivalent to *True*.
  - From our definition of entailment, we can derive the **deduction theorem**, which was known to the ancient Greeks:
    - *For any sentences α and β, $\alpha \models \beta$ if and only if the sentence $(\alpha \Rightarrow \beta)$ is valid.*

5. Propositional Theorem Proving

- **Satisfiablility**: A sentence is satisfiable if it is true in, or satisfied by, *some* model.
  - For example, the knowledge base given earlier, $(R_1 \wedge R_2 \wedge R_3 \wedge R_4 \wedge R_5)$, is satisfiable because there are three models in which it is true.
  - Satisfiability can be checked by enumerating the possible models until one is found that satisfies the sentence.
  - The problem of determining the satisfiability of sentences in propositional logic—the **SAT** problem—was the first problem proved to be NP-complete. Many problems in computer science are really satisfiability problems.
- Validity and satisfiability are connected: α is valid iff $\neg\alpha$ is unsatisfiable; contrapositively, α is satisfiable iff $\neg\alpha$ is not valid.
  - *$\alpha \models \beta$ if and only if the sentence $(\alpha \wedge \neg\beta)$ is unsatisfiable.*
    - Proving β from α by checking the unsatisfiability of $(\alpha \wedge \neg\beta)$ corresponds exactly to the standard mathematical proof technique of reductio ad absurdum. It is also called proof by **refutation** or proof by **contradiction**. One assumes a sentence β to be false and shows that this leads to a contradiction with known axioms α.

5. Propositional Theorem Proving

Inference and Proofs

- Inference rules can be applied to derive a **proof**.
  - A proof is a chain of conclusions that leads to the desired goals.
- **Modus Ponens** $\frac{\propto \Rightarrow \beta, \alpha}{\beta}$
  - The notation means that, whenever any sentences of the form $\alpha \Rightarrow \beta$ and α are given, then the sentence β can be inferred.
- **And-Elimination** $\frac{\alpha \wedge \beta}{\alpha}$
  - From a conjunction, any of the conjuncts can be inferred.
- By considering the possible truth values of α and β, one can easily show once and for all 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.

5. Propositional Theorem Proving

Inference and Proofs

- Searching for proofs is an alternative to enumerating models.
- In many practical cases *finding a proof can be more efficient because the proof can ignore irrelevant propositions, no matter how many of them there are.*

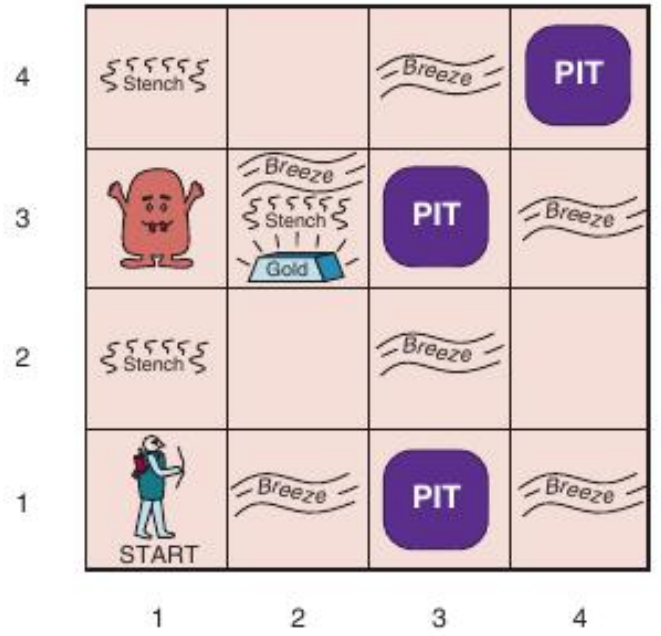
5. Propositional Theorem Proving
Inference and Proofs

- **Monotonicity**
  - The set of entailed sentences can only *increase* as information is added to the knowledge base.
  - For any sentences α and β, 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.

5. Propositional Theorem Proving

Inference and Proofs

- Let us see how these inference rules and equivalences can be used in the wumpus world.
- We start with the knowledge base containing R_1 through R_5 and show how to prove $\neg P_{1,2}$, that is, there is no pit in [1,2]:

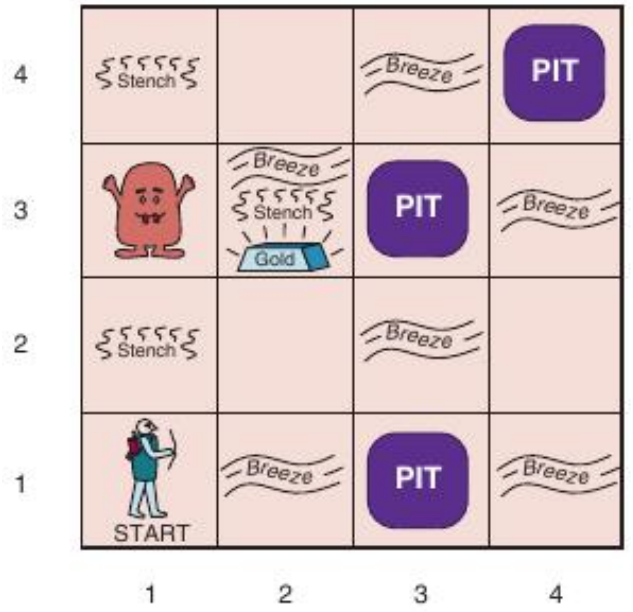


The knowledge base (R_1 through R_5):

- R_1: $\neg P_{1,1}$.
- R_2: $B_{1,1} \Leftrightarrow (P_{1,2} \vee P_{2,1})$.
- R_3: $B_{2,1} \Leftrightarrow (P_{1,1} \vee P_{2,2} \vee P_{3,1})$.
- R_4: $\neg B_{1,1}$.
- R_5: $B_{2,1}$.

5. Propositional Theorem Proving
Inference and Proofs

- To prove $\neg P_{1,2}$, that is, there is no pit in [1,2]:
  - 1. Apply biconditional elimination to R_2 to obtain
    - $R_6 : (B_{1,1} \Rightarrow (P_{1,2} \vee P_{2,1})) \wedge ((P_{1,2} \vee P_{2,1}) \Rightarrow B_{1,1})$.
  - 2. Apply And-Elimination to R_6 to obtain
    - $R_7 : ((P_{1,2} \vee P_{2,1}) \Rightarrow B_{1,1})$.
  - 3. Logical equivalence for contrapositives gives
    - $R_8 : (\neg B_{1,1} \Rightarrow \neg(P_{1,2} \vee P_{2,1}))$.
  - 4. Apply Modus Ponens with R_8 and the percept R_4 (i.e., $\neg B_{1,1}$), to obtain
    - $R_9 : \neg(P_{1,2} \vee P_{2,1})$.
  - 5. Apply De Morgan's rule, giving the conclusion
    - $R_{10} : \neg P_{1,2} \wedge \neg P_{2,1}$.



The knowledge base (R_1 through R_5):

- R_1: $\neg P_{1,1}$.
- R_2: $B_{1,1} \Leftrightarrow (P_{1,2} \vee P_{2,1})$.
- R_3: $B_{2,1} \Leftrightarrow (P_{1,1} \vee P_{2,2} \vee P_{3,1})$.
- R_4: $\neg B_{1,1}$.
- R_5: $B_{2,1}$.

*

5. Propositional Theorem Proving
Inference and Proofs

- Any of the search algorithms in Chapter 3 can be used to find a sequence of steps that constitutes a proof like this. 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.
- Thus, searching for proofs is an alternative to enumerating models. In many practical cases *finding a proof can be more efficient because the proof can ignore irrelevant propositions, no matter how many of them there are*.

5. Propositional Theorem Proving

Proof by Resolution

- Search algorithms 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.
- The current section introduces a single inference rule, **resolution**, that yields a complete inference algorithm when coupled with any complete search algorithm.
- We begin by using a simple version of the resolution rule in the wumpus world.

5. Propositional Theorem Proving
Proof by Resolution

- Let us consider the steps leading up to Figure 7.4(a): the agent returns from [2,1] to [1,1] and then goes to [1,2], where it perceives a stench, but no breeze.
  - We add the following facts to the knowledge base:
    - $R_{11} : \neg B_{1,2}$.
    - $R_{12} : B_{1,2} \Leftrightarrow (P_{1,1} \vee P_{2,2} \vee P_{1,3})$.
  - We can now derive the absence of pits in [2,2] and [1,3]:
    - $R_{13} : \neg P_{2,2}$.
    - $R_{14} : \neg P_{1,3}$.
  - R_3, followed by Modus Ponens with R_5:
    - $R_{15} : P_{1,1} \vee P_{2,2} \vee P_{3,1}$.
  - The literal $\neg P_{2,2}$ in R_{13} *resolves with* the literal $P_{2,2}$ in R_{15} to give the **resolvent**:
    - $R_{16} : P_{1,1} \vee P_{3,1}$.
  - Similarly, the literal $\neg P_{1,1}$ in R_1 resolves with the literal $P_{1,1}$ in R_{16} to give
    - $R_{17} : P_{3,1}$.

Unit resolution inference rule

1,4	2,4	3,4	4,4
1,3 W!	2,3	3,3	4,3
1,2 A S OK	2,2 OK	3,2	4,2
1,1 V OK	2,1 B V OK	3,1 P!	4,1

The knowledge base:

- R_1: $\neg P_{1,1}$.
- R_2: $B_{1,1} \Leftrightarrow (P_{1,2} \vee P_{2,1})$.
- R_3: $B_{2,1} \Leftrightarrow (P_{1,1} \vee P_{2,2} \vee P_{3,1})$.
- R_4: $\neg B_{1,1}$.
- R_5: $B_{2,1}$.
- R_6: $(B_{1,1} \Rightarrow (P_{1,2} \vee P_{2,1})) \wedge ((P_{1,2} \vee P_{2,1}) \Rightarrow B_{1,1})$.
- R_7: $((P_{1,2} \vee P_{2,1}) \Rightarrow B_{1,1})$.
- R_8: $(\neg B_{1,1} \Rightarrow \neg(P_{1,2} \vee P_{2,1}))$.
- R_9: $\neg(P_{1,2} \vee P_{2,1})$.
- R_{10}: $\neg P_{1,2} \wedge \neg P_{2,1}$.

5. Propositional Theorem Proving
Proof by Resolution

- The unit resolution inference rule:

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

  - each ℓ is a literal and ℓ_i and m are **complementary literals** (i.e., one is the negation of the other). Thus, the unit resolution rule takes a **clause**—a disjunction of literals—and a literal and produces a new clause.

- Full resolution rule:

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

  - 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. For example:

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

5. Propositional Theorem Proving

Proof by Resolution

- We can resolve only one pair of complementary literals at a time. For example, we can resolve P and $\neg P$ to deduce

$$\frac{P \vee \neg Q \vee R, \qquad \neg P \vee Q}{\neg Q \vee Q \vee R},$$

  - We can't resolve on both P and Q at once to infer R. There is one more technical aspect of the 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 $(A \vee B)$ with $(A \vee \neg B)$, we obtain $(A \vee A)$, which is reduced to just A by factoring.

5. Propositional Theorem Proving

Proof by Resolution

- What is more surprising about the resolution rule is that it forms the basis for a family of *complete* inference procedures.
- *A resolution-based theorem prover can, for any sentences α and β in propositional logic, decide whether α |= β.*

5. Propositional Theorem Proving

Proof by Resolution

Conjunctive Normal Form (CNF)

- The resolution rule applies only to clauses that is, disjunctions of literals.
- How, then, can it lead to a complete inference procedure for all of propositional logic?
- The answer is that *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**.

5. Propositional Theorem Proving

Proof by Resolution

Conjunctive Normal Form (CNF)

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

Figure 7.12 A grammar for conjunctive normal form, Horn clauses, and definite clauses. A CNF clause such as $\neg A \vee \neg B \vee C$ can be written in definite clause form as $A \wedge B \Rightarrow C$.

5. Propositional Theorem Proving

Proof by Resolution

Conjunctive Normal Form (CNF)

- Converting the sentence $B_{1,1} \Leftrightarrow (P_{1,2} \vee P_{2,1})$ into CNF:
  - 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. CNF requires $\neg$ to appear only in literals, so we "move $\neg$ inwards" by repeated application of the following equivalences:
    - $\neg(\neg\alpha) \equiv \alpha$ (double-negation 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)

    In the example, we require just one application of the last rule:
    - $(\neg B_{1,1} \vee P_{1,2} \vee P_{2,1}) \wedge ((\neg P_{1,2} \wedge \neg P_{2,1}) \vee B_{1,1})$.
  - 4. 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.
    - $(\neg B_{1,1} \vee P_{1,2} \vee P_{2,1}) \wedge (\neg P_{1,2} \vee B_{1,1}) \wedge (\neg P_{2,1} \vee B_{1,1})$.

    The original sentence is now in CNF, as a conjunction of three clauses. It is much harder to read, but it can be used as input to a resolution procedure.

5. Propositional Theorem Proving

Proof by Resolution

A Resolution Algorithm

- Inference procedures based on resolution work by using the principle of proof by contradiction.
  - $KB \models \alpha$ if and only if the sentence $(KB \wedge \neg\alpha)$ is unsatisfiable.
- A resolution algorithm:
  - 1. $(KB \wedge \neg\alpha)$ is converted into CNF.
  - 2. 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; or,
    - Two clauses resolve to yield the *empty* clause, in which case *KB* entails α.
      - The empty clause—a disjunction of no disjuncts—is equivalent to *False* because a disjunction is true only if at least one of its disjuncts is true. Moreover, the empty clause arises only from resolving two contradictory unit clauses such as P and $\neg P$.

5. Propositional Theorem Proving

Proof by Resolution

A Resolution Algorithm

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 $\leftarrow \{\}$
  while *true* **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*

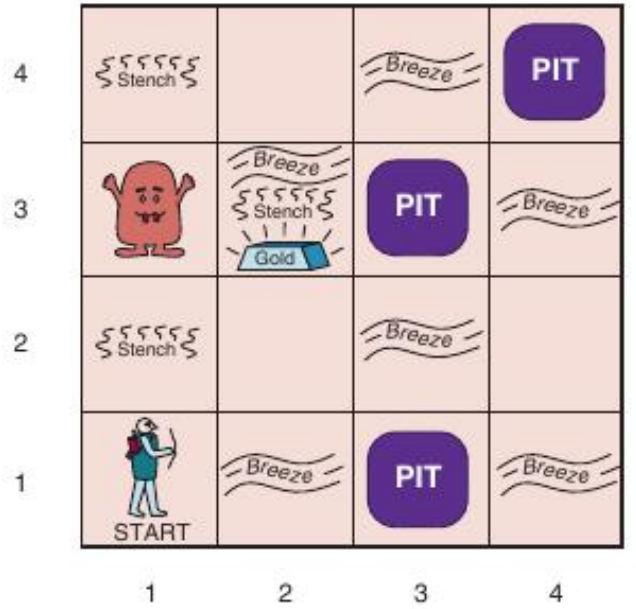
Figure 7.13 A simple resolution algorithm for propositional logic. PL-RESOLVE returns the set of all possible clauses obtained by resolving its two inputs.

5. Propositional Theorem Proving

Proof by Resolution

A Resolution Algorithm

- We can apply the resolution procedure to a very simple inference in the wumpus world.
- 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
  - $KB = R_2 \wedge R_4 = (B_{1,1} \Leftrightarrow (P_{1,2} \vee P_{2,1})) \wedge \neg B_{1,1}$
- We wish to prove $KB \models \alpha$, $\alpha = \neg P_{1,2}$.
  - Convert $(KB \wedge \neg\alpha)$ into CNF: Clauses shown at the first row of Figure 7.14.
  - The second row of the figure shows clauses obtained by resolving pairs in the first row.



The knowledge base (R_1 through R_5):

- R_1: $\neg P_{1,1}$.
- R_2: $B_{1,1} \Leftrightarrow (P_{1,2} \vee P_{2,1})$.
- R_3: $B_{2,1} \Leftrightarrow (P_{1,1} \vee P_{2,2} \vee P_{3,1})$.
- R_4: $\neg B_{1,1}$.
- R_5: $B_{2,1}$.

5. Propositional Theorem Proving

Proof by Resolution

A Resolution Algorithm

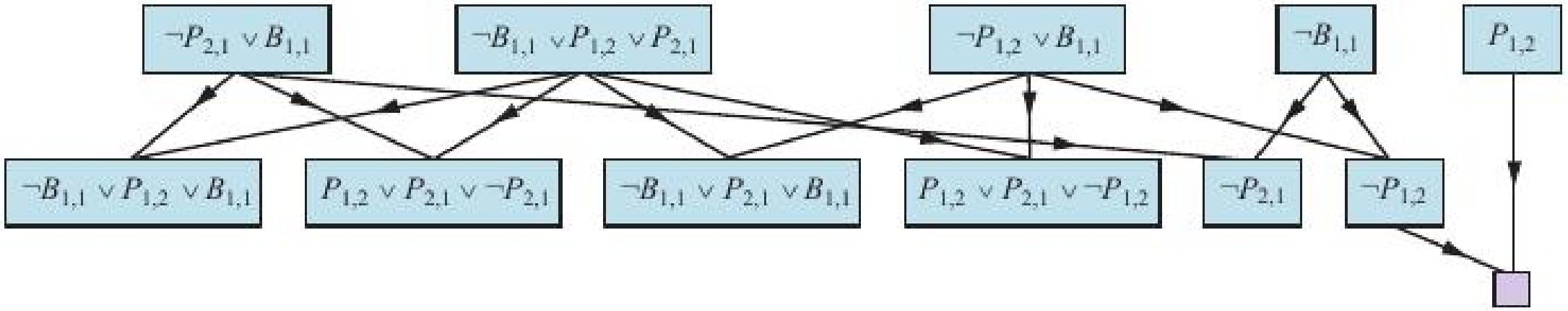


Figure 7.14 Partial application of PL-RESOLUTION to a simple inference in the wumpus world to prove the query $\neg P_{1,2}$. Each of the leftmost four clauses in the top row is paired with each of the other three, and the resolution rule is applied to yield the clauses on the bottom row. We see that the third and fourth clauses on the top row combine to yield the clause $\neg P_{1,2}$, which is then resolved with $P_{1,2}$ to yield the empty clause, meaning that the query is proven.

5. Propositional Theorem Proving

Proof by Resolution

Completeness of Resolution

- Why is PL-R ESOLUTION complete?
- **Resolution closure** *RC*(*S*) of a set of clauses *S:* is the set of all clauses derivable by repeated application of the resolution rule to clauses in *S* or their derivatives.
- The resolution closure is what PL-R ESOLUTION computes as the final value of the variable *clauses*.
- It is easy to see that *RC*(*S*) must be finite: thanks to the factoring step, there are only finitely many distinct clauses that can be constructed out of the symbols P_1, ..., P_k that appear in *S*. Hence, PL-RESOLUTION always terminates.

5. Propositional Theorem Proving

Proof by Resolution

Completeness of Resolution

- The completeness theorem for resolution in propositional logic is called the **ground resolution** theorem:
  - *If a set of clauses is unsatisfiable, then the resolution closure of those clauses contains the empty clause.*

5. Propositional Theorem Proving
Horn Clauses and Definition Clauses

- The completeness of resolution makes it a very important inference method. In many practical situations, however, the full power of resolution is not needed.
- Some real-world knowledge bases satisfy certain restrictions on the form of sentences they contain, which enables them to use a more restricted and efficient inference algorithm.
- A **Horn clause**: is a disjunction of literals of which *at most one is positive*.
  - A **definite clause**: which is a disjunction of literals of which *exactly one is positive*.
    - For example, the clause $(\neg L_{1,1} \vee \neg Breeze \vee B_{1,1})$ is a definite clause, whereas $(\neg B_{1,1} \vee P_{1,2} \vee P_{2,1})$ is not, because it has two positive clauses.
  - A **goal clause**: which is a disjunction of literals with *no positive literals*.
- Horn clauses are closed under resolution: *if you resolve two Horn clauses, you get back a Horn clause*.

5. Propositional Theorem Proving
Horn Clauses and Definition 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.
    - $(\neg L_{1,1} \vee \neg Breeze \vee B_{1,1}) \equiv \quad (L_{1,1} \wedge Breeze) \Rightarrow B_{1,1}$
    - In the implication form, the sentence is easier to understand: it says that if the agent is in [1,1] and there is a breeze percept, then [1,1] is breezy.
    - In Horn form, the premise is called the **body** and the conclusion is called the **head**.
    - A **fact**: 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 $L_{1,1}$.
  - 2. Inference with Horn clauses can be done through the **forward-chaining** and **backward-chaining** algorithms. Both of these algorithms are natural, in that the inference steps are obvious and easy for humans to follow. 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—a pleasant surprise.