Knowledge-based agents are best understood as agents that know about their world and reason about their courses of action.

Basic concepts:

- The knowledge-base (KB): a set of representations of facts about the world.

- The knowledge representation language: a language whose sentences represent facts about the world.
Knowledge-Based Agents (cont’d)

- **TELL** and **ASK** interface: operations for adding new sentences to the KB and querying what is known. This is similar to updating and querying in databases.

- The **inference mechanism**: a mechanism for determining what follows from what has been TELLed to the knowledge base. The ASK operation utilizes this inference mechanism.
A Generic 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
We can describe a knowledge-based agent at three levels:

- **The knowledge level:** In this level the agent is specified by saying what it knows about the world and what its goals are.
- **The logical level:** This is the level at which the knowledge is encoded into sentences of some logical language.
- **The implementation level:** This is the level where sentences are implemented. This level runs on the agent architecture.

**Note:** Declarative vs. **procedural** way of system building
Knowledge-based Agents (cont’d)

Example:

- **Knowledge level or epistemological level:**
  The automated taxi driver knows that Golden Gate Bridge links San Francisco and Marin County.

- **Logical level:**
  The automated taxi driver has the FOL sentence \( Links(GGBridge, SF, Marin) \) in its KB.

- **Implementation level:**
  The sentence \( Links(GGBridge, SF, Marin) \) is implemented by a C structure (or a Prolog fact).
We can build a knowledge-based agent by TELLing it what it needs to know before it starts perceiving the world.

We can also design learning mechanisms that output general knowledge about the environment given a series of percepts.

Autonomous agent = Knowledge-based agent + Learning mechanism
The Wumpus World (WW)

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>Stench</td>
<td>Breeze</td>
<td>PIT</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Breeze</td>
<td>PIT</td>
<td>Breeze</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Stench</td>
<td>Gold</td>
<td>Breeze</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>START</td>
<td>Breeze</td>
<td>PIT</td>
<td>Breeze</td>
</tr>
</tbody>
</table>

1 2 3 4
• **Environment:** 4x4 grid of rooms with agent, wumpus, gold and pits.

• **Actuators:** The agent can move forward, turn left or turn right. The agent dies if it enters a room with a pit or a live wumpus.

  The agent has action *Grab* and *Shoot* (one arrow only) at its disposal.

• **Sensors:** The percept is a list of 5 symbols:

  \[(Stench, Breeze, Glitter, Bump, Scream)\]

  Any of the above values can be *None*.  

The WW (cont’d)
Reasoning and Acting in the WW

<table>
<thead>
<tr>
<th></th>
<th>2,1</th>
<th>3,1</th>
<th>4,1</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>A</strong></td>
<td>OK</td>
<td>OK</td>
<td>OK</td>
</tr>
<tr>
<td>1,1</td>
<td>V</td>
<td>B</td>
<td>OK</td>
</tr>
</tbody>
</table>

(a)

<table>
<thead>
<tr>
<th></th>
<th>2,1</th>
<th>3,4</th>
<th>4,4</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>A</strong></td>
<td>Agent</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>B</strong></td>
<td>Breeze</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>G</strong></td>
<td>Glitter, Gold</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>OK</strong></td>
<td>Safe square</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>P</strong></td>
<td>Pit</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>S</strong></td>
<td>Stench</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>V</strong></td>
<td>Visited</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>W</strong></td>
<td>Wumpus</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

(b)
### Reasoning and Acting in the WW (cont’d)

<table>
<thead>
<tr>
<th></th>
<th>2,4</th>
<th>3,4</th>
<th>4,4</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>B</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>G</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>OK</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>P</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>S</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>V</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>W</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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

![Diagram](image)

(a) (b)
Reasoning and Acting in the WW (cont’d)

What do we need?

- A way to **learn facts** about the world.
- A way to **represent facts** and **rules** about the world.
- A way to **reason** with the existing facts in order to **deduce new ones**.

Examples of facts:

- The wumpus is in square [1,3].
- There is **no** pit in square [2,2].
- There is a pit in square [2,2] **or** square [3,1].
A KR language is defined by specifying its **syntax** and **semantics**. The **syntax** of a KR language specifies the well-formed formulas and sentences. The **semantics** of a KR language defines a correspondence between formulas/sentences of the language and facts in the world to which these formulas/sentences refer. A sentence of a KR language does not mean anything by itself. The semantics or **meaning** of a sentence must be provided by its writer by means of an **interpretation**.
**Truth and Entailment**

**Truth.** A sentence will be called true under a particular interpretation if the state of affairs it represents is the case.

**Entailment.** We will write $KB \models \alpha$ to denote that whenever the sentences of $KB$ are true, then the sentence $\alpha$ is also true. In this case we will say that the sentences of $KB$ entail the sentence $\alpha$.

Given a knowledge-base $KB$ and a sentence $\alpha$, how do we design an algorithm that verifies whether $KB \models \alpha$?
Entailment (cont’d)

Representations

World

Semantics

Facts

Follows

Sentence

Entails

Sentence

Facts

Follows

Fact
Inference is the process of mechanically deriving sentences entailed by a knowledge-base. If sentence $\alpha$ is derived from $KB$ using inference mechanism $i$ then we will write $KB \vdash_i \alpha$.

An inference mechanism is called **sound** if it derives only sentences that are entailed.

An inference mechanism is called **complete** if it derives all the sentences that are entailed.

The steps used to derive a sentence $\alpha$ from a set of sentences $KB$ is called a **proof**.

A **proof theory** is a set of rules for deriving the entailments of a set of sentences.
The KR languages we will consider will be based on **propositional logic** and **first-order logic**.

Thus, the knowledge-based agents we will design can also be called **logical agents**.

In general, a **logic** is a formal system consisting of:

- Syntax
- Semantics
- Proof theory

**Question:** Why do we use logical languages for KR? Why don’t we use natural language or programming languages?
Propositional Logic (PL): Syntax

The symbols of PL are:

- A countably infinite set of proposition symbols $P_1, P_2, \ldots$. This set will be denoted by $\mathcal{P}$.
- The logical connectives $\neg, \land, \lor, \Rightarrow$ and $\Leftrightarrow$.
- Parentheses: $\left(, \right)$.

Note: Logicians usually introduce only the connectives $\neg$ and $\lor$ and define the rest in terms of them.
The following context-free grammar defines the well-formed sentences of propositional logic.

\[
\begin{align*}
\text{Sentence} & \rightarrow \text{AtomicSentence} \mid \text{ComplexSentence} \\
\text{AtomicSentence} & \rightarrow \text{True} \mid \text{False} \mid \text{Symbol} \\
\text{Symbol} & \rightarrow P_1 \mid P_2 \mid \cdots \\
\text{ComplexSentence} & \rightarrow (\text{Sentence}) \mid \neg \text{Sentence} \\
& \quad \mid \text{Sentence BinaryConnective Sentence} \\
\text{BinaryConnective} & \rightarrow \land \mid \lor \mid \Rightarrow \mid \Leftrightarrow \\
\end{align*}
\]

Precedence: \(\neg, \land, \lor, \Rightarrow \) and \(\Leftrightarrow\).
A proposition symbol can mean anything we want. Its interpretation can be any arbitrary fact. This fact will be either true or false in the world. This is not the same in other logics (e.g., fuzzy logic!).

This is formalized by introducing the notion of interpretation.

**Definition.** Let \( \mathcal{P} \) be the set of proposition symbols. An interpretation for \( \mathcal{P} \) is a mapping

\[
I : \mathcal{P} \rightarrow \{true, false\}.
\]
The notion of interpretation can be extended to arbitrary well-formed sentences of PL using the following recursive definitions:

- \( I(True) = true \).
- \( I(False) = false \).
- \( I(\neg \phi) = true \) if \( I(\phi) = false \); otherwise it is \( false \).
- \( I(\phi_1 \land \phi_2) = true \) if \( I(\phi_1) = true \) and \( I(\phi_2) = true \); otherwise it is \( false \).
- \( I(\phi_1 \lor \phi_2) = true \) if \( I(\phi_1) = true \) or \( I(\phi_2) = true \); otherwise it is \( false \).
• \( I(\phi_1 \Rightarrow \phi_2) = \text{true} \) if \( I(\phi_1) = \text{false} \) or \( I(\phi_2) = \text{true} \); otherwise it is \( \text{false} \).

**Explanation:** If \( \phi_1 \) and \( \phi_2 \) are both true then most people would agree that \( \phi_1 \Rightarrow \phi_2 \) (\( \phi_1 \) implies \( \phi_2 \)) should be true.

**Example:** For all integers, if \( x \) is even then \( x + 2 \) is even. If we take \( x \) to be 6 then this says: “If 6 is even then 6+2 is even”. But what about cases where the truth value of \( \phi_1 \) is \( \text{false} \)?

**Example:** If we take \( x \) to be 7 then the above formula says: “If 7 is even then 7+2 is even”. Is this sentence true or false? This is an instance of a “false implies false” implication.
We will take the above sentence to be true although some of us might find it disconcerting. It would be wrong to take it to be false given that the more general sentence of which it is an instance is true.

We have similar difficulties for “false implying true”.

Example: If 1+1=3 then Athens is the capital of Greece.

The case “true implying false” is easier: most people would accept such an implication to be false.

Thus we have taken “implication” to have the semantics of material implication.
• \( I(\phi_1 \Leftrightarrow \phi_2) = true \) if \( I(\phi_1) = I(\phi_2) \); otherwise it is \( false \).
A language is called *compositional* when the meaning of a sentence is a function of the meaning of the parts.

Compositionality is a desirable property in formal languages.
The Ontological Commitments of PL

Ontological commitments have to do with the nature of reality. PL assumes that the world consists of facts that either hold or not hold.

Other logics, for example FOL, make more elaborate and detailed ontological commitments.
Satisfaction and Models

**Definition.** Let $\phi$ be a PL sentence. If $I$ is an interpretation such that $I(\phi) = true$ then we say that $I$ satisfies $\phi$ or $I$ is a model of $\phi$. 
**Satisfiability**

**Definition.** A sentence $\phi$ of PL is **satisfiable** if there is an interpretation $I$ such that $I(\phi) = true$.

**Examples:** $P, P \lor Q, (P \land R) \lor Q$

**Definition.** A sentence $\phi$ of PL is **unsatisfiable** if there is no interpretation $I$ such that $I(\phi) = true$.

**Example:** $P \land \neg P$
Validity

**Definition.** A sentence $\phi$ of PL is **valid** if for all interpretations $I$, $I(\phi) = true$.

**Examples:** $P \lor \neg P$, $((P \lor H) \land \neg H) \Rightarrow P$

Valid statements in PL are also called **tautologies**.

**Theorem.** Let $\phi$ be a sentence of PL. If $\phi$ is unsatisfiable then its negation $\neg \phi$ is valid. Proof?
Entailment

**Definition.** Let $\phi$ and $\psi$ be sentences of PL. We will say that $\phi$ entails $\psi$ or $\phi$ logically implies $\psi$ (denoted by $\phi \models \psi$) if for all interpretations $I$ such that $I(\phi) = true$ then $I(\psi) = true$.

**Example:** $P \land Q \models P$

**The deduction theorem.** Let $\phi$ and $\psi$ be sentences of PL. Then $\phi \models \psi$ iff $\phi \Rightarrow \psi$ is valid. Proof?

**Example:** $(P \land Q) \Rightarrow P$ is a valid sentence.
**Entailment and Unsatisfiability**

**Theorem.** Let $\phi$ and $\psi$ be sentences of PL. Then $\phi \models \psi$ iff $\phi \land \neg \psi$ is unsatisfiable. Proof?

**Example:** $P \land Q \models P$

The above theorem is the essence of **proofs by contradiction** or **refutation**.
Definition. Let $\phi$ and $\psi$ be sentences of PL. We will say that $\phi$ is equivalent to $\psi$ (denoted by $\phi \equiv \psi$) if $\phi \models \psi$ and $\psi \models \phi$.

Example: $\neg(P \land \neg Q) \equiv \neg P \lor Q$
Some Useful Equivalences

- $(\alpha \land \beta) \equiv (\beta \land \alpha)$  commutativity of $\land$
- $(\alpha \lor \beta) \equiv (\beta \lor \alpha)$  commutativity of $\lor$
- $((\alpha \land \beta) \land \gamma) \equiv (\alpha \land (\beta \land \gamma))$  associativity of $\land$
- $((\alpha \lor \beta) \lor \gamma) \equiv (\alpha \lor (\beta \lor \gamma))$  associativity of $\lor$
- $\neg(\neg \alpha) \equiv \alpha$  double-negation elimination
- $(\alpha \Rightarrow \beta) \equiv (\neg \beta \Rightarrow \neg \alpha)$  contraposition
Some Useful Equivalences (cont’d)

- \((\alpha \Rightarrow \beta) \equiv (\neg \alpha \vee \beta)\) implication elimination
- \((\alpha \iff \beta) \equiv ((\alpha \Rightarrow \beta) \land (\beta \Rightarrow \alpha))\) biconditional elimination
- \((\neg (\alpha \land \beta)) \equiv (\neg \alpha \lor \neg \beta)\) de Morgan law
- \((\neg (\alpha \lor \beta)) \equiv (\neg \alpha \land \neg \beta)\) de Morgan law
- \((\alpha \land (\beta \lor \gamma)) \equiv ((\alpha \land \beta) \lor (\alpha \land \gamma))\) distribution of \(\land\) over \(\lor\)
- \((\alpha \lor (\beta \land \gamma)) \equiv ((\alpha \lor \beta) \land (\alpha \lor \gamma))\) distribution of \(\lor\) over \(\land\)
**Truth Tables**

Truth tables are tools that allow us to find the truth value of a PL formula if we know the truth value of its constituent parts.

<table>
<thead>
<tr>
<th>$A$</th>
<th>$\neg A$</th>
</tr>
</thead>
<tbody>
<tr>
<td>true</td>
<td>false</td>
</tr>
<tr>
<td>false</td>
<td>true</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$A$</th>
<th>$B$</th>
<th>$A \land B$</th>
<th>$A \lor B$</th>
<th>$A \Rightarrow B$</th>
<th>$A \Leftrightarrow B$</th>
</tr>
</thead>
<tbody>
<tr>
<td>false</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>true</td>
<td>true</td>
</tr>
<tr>
<td>false</td>
<td>true</td>
<td>false</td>
<td>true</td>
<td>true</td>
<td>false</td>
</tr>
<tr>
<td>true</td>
<td>false</td>
<td>false</td>
<td>true</td>
<td>false</td>
<td>false</td>
</tr>
<tr>
<td>true</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>true</td>
</tr>
</tbody>
</table>
Truth Tables

Why are truth tables useful?

Truth tables can be used for showing the truth or falsity of any PL sentence in a given interpretation.

Similarly, truth tables can be used for showing the satisfiability or validity of any PL sentence.
**Example:** A truth table for showing the validity of sentence 
\(((P \lor H) \land \neg H) \Rightarrow P\). 

<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>$P$</td>
<td>$H$</td>
<td>$P \lor H$</td>
<td>$(P \lor H) \land \neg H$</td>
<td>$((P \lor H) \land \neg H)$</td>
<td>$\Rightarrow P$</td>
</tr>
<tr>
<td><strong>false</strong></td>
<td><strong>false</strong></td>
<td><strong>false</strong></td>
<td><strong>false</strong></td>
<td><strong>true</strong></td>
<td></td>
</tr>
<tr>
<td><strong>false</strong></td>
<td><strong>true</strong></td>
<td><strong>true</strong></td>
<td><strong>false</strong></td>
<td><strong>true</strong></td>
<td></td>
</tr>
<tr>
<td><strong>true</strong></td>
<td><strong>false</strong></td>
<td><strong>true</strong></td>
<td><strong>true</strong></td>
<td><strong>true</strong></td>
<td></td>
</tr>
<tr>
<td><strong>true</strong></td>
<td><strong>true</strong></td>
<td><strong>true</strong></td>
<td><strong>false</strong></td>
<td><strong>true</strong></td>
<td></td>
</tr>
</tbody>
</table>
PL Satisfiability as a CSP

The satisfiability problem for PL is fundamental. The entailment and validity problems can be rephrased as satisfiability problems.

Notice that the satisfiability problem for PL can be phrased as a CSP. What are the variables, values and constraints?
Complexity of PL Satisfiability and Validity

**Theorem.** The problem of determining whether a sentence of PL is satisfiable is NP-complete (Cook, 1971).

**Corollary.** The problem of determining whether a sentence of PL is valid is co-NP-complete.

The above results mean that it is highly unlikely that we will ever find a polynomial time algorithm for these problems.
Horn Sentences

Definition. A PL sentence will be called **Horn** if it is in one of the following forms:

\[
P_1 \land P_2 \land \ldots \land P_n \Rightarrow Q \quad \text{or equivalently} \quad \neg P_1 \lor \neg P_2 \lor \ldots \lor \neg P_n \lor Q
\]

\[

\neg P_1 \lor \neg P_2 \lor \ldots \lor \neg P_n
\]

Theorem. If \( \phi \) is a conjunction of Horn sentences then the satisfiability of \( \phi \) can be decided in polynomial time.
Inference

Let $KB$ be a set of PL sentences and $\phi$ a PL sentence. How can we decide whether $KB \models \phi$?

Use truth tables! (either explicitly or by writing an appropriate algorithm)

**Computational complexity:** $O(2^n)$ where $n$ is the number of propositional symbols in $KB$ and $\phi$. 
Example

A truth table showing that \( \{ P \lor H, \neg H \} \models P \).

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>( P )</td>
<td>( H )</td>
<td>( P \lor H )</td>
<td>( \neg H )</td>
<td>( P )</td>
</tr>
<tr>
<td>false</td>
<td>false</td>
<td>false</td>
<td>true</td>
<td>false</td>
</tr>
<tr>
<td>false</td>
<td>true</td>
<td>true</td>
<td>false</td>
<td>false</td>
</tr>
<tr>
<td>true</td>
<td>false</td>
<td>true</td>
<td>true</td>
<td>true</td>
</tr>
<tr>
<td>true</td>
<td>true</td>
<td>true</td>
<td>false</td>
<td>true</td>
</tr>
</tbody>
</table>

Method: Go through all the lines of the truth table that make each sentence of \( KB \) true, and check whether they make \( \phi \) true too.
An inference rule is a rule of the form

\[
\frac{\alpha_1, \alpha_2, \ldots, \alpha_n}{\beta}
\]

where \(\alpha_1, \alpha_2, \ldots, \alpha_n\) are sentences called conditions and \(\beta\) is a sentence called conclusion.

Whenever we have a set of sentences that match the conditions of an inference rule then we can conclude the sentence in the conclusion.
Inference Rules for PL (cont’d)

- **Modus Ponens:** \( \frac{\alpha \Rightarrow \beta, \alpha}{\beta} \)
- **And-Elimination:** \( \frac{\alpha_1 \land \alpha_2 \land \ldots \land \alpha_n}{\alpha_i} \)
- **And-Introduction:** \( \frac{\alpha_1, \alpha_2, \ldots, \alpha_n}{\alpha_1 \land \alpha_2 \land \ldots \land \alpha_n} \)
- **Or-Introduction:** \( \frac{\alpha_i}{\alpha_1 \lor \alpha_2 \lor \ldots \lor \alpha_n} \)
- **Double-Negation Elimination:** \( \frac{\neg \neg \alpha}{\alpha} \)
- **Unit Resolution:** \( \frac{\alpha \lor \beta, \neg \beta}{\alpha} \)
- **Resolution:** \( \frac{\alpha \lor \beta, \neg \beta \lor \gamma}{\alpha \lor \gamma} \)
Why Is Inference Important?
Formalizing the WW in PL

<table>
<thead>
<tr>
<th></th>
<th>2,1</th>
<th>3,1</th>
<th>4,1</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>OK</td>
<td>OK</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>2,2</th>
<th>3,2</th>
<th>4,2</th>
</tr>
</thead>
<tbody>
<tr>
<td>OK</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>2,3</th>
<th>3,3</th>
<th>4,3</th>
</tr>
</thead>
<tbody>
<tr>
<td>B</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>2,4</th>
<th>3,4</th>
<th>4,4</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Formalizing the WW in PL (cont’d)

The WW **knowledge base** for Figure (a):

- There is no pit in \([1, 1]\) (agent percept):

  \[ R_1 : \neg P_{11} \]

- A square is breezy if and only if there is a pit in a neighboring square. (Rule of the WW). We state this for the square \(B_{11}\) only:

  \[ R_2 : \quad B_{11} \Leftrightarrow P_{12} \lor P_{21} \]

- There is no breeze in square \([1, 1]\). (agent percept)

  \[ R_3 : \quad \neg B_{11} \]
The agent can now use the **PL inference rules** and **logical equivalences** to prove the following: “There is no pit in squares [1, 2] or [1, 2]”

- Apply biconditional elimination to $R_2$:

  \[ R_4 : (B_{11} \Rightarrow (P_{12} \lor P_{21})) \land ((P_{12} \lor P_{21}) \Rightarrow B_{11}) \]

- Apply And-elimination to $R_4$:

  \[ R_5 : (P_{12} \lor P_{21}) \Rightarrow B_{11} \]

- Apply logical equivalence for contrapositives to $R_5$:

  \[ R_6 : \neg B_{11} \Rightarrow \neg(P_{12} \lor P_{21}) \]
Inference in the WW (cont’d)

• Apply modus ponens to $R_6$ and $R_3$:

$$R_7 : \neg(P_{12} \lor P_{21})$$

• Apply de Morgan’s rule to $R_7$:

$$R_8 : \neg P_{12} \land \neg P_{21}$$
Consider now the following WW rule:

If a square has no smell, then neither the square nor any of its adjacent squares can house a Wumpus.

How can we formalize this rule in PL?

We have to write **one rule for every relevant square**! For example:

\[ \neg S_{11} \Rightarrow \neg W_{11} \land \neg W_{12} \land W_{21} \]

This is a very disappointing feature of PL. **There is no way in PL to make a statement referring to all objects of some kind** (e.g., to all squares).

Not to worry: this can be done in first order logic!
A knowledge-based agent using PL

function PROPOSITIONAL-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))

for each action in the list of possible actions do

    if ASK(KB,MAKE-ACTION-QUERY(t, action)) then

        TELL(KB,MAKE-ACTION-SENTENCE(action, t))

        t ← t + 1

    return action

end
Readings

Chapter 7 of AIMA: Logical Agents (Sections 7.1 to 7.5 and 7.7)

We did not cover all the material in Section 7.5 in great detail (e.g., resolution) because these techniques will be revisited in the more general framework of FOL in the forthcoming lectures.