1 Propositional Logic and AI

Artificial Intelligence aims to create agents that not only search for solutions but "understand" the world. While search algorithms generate successors and evaluate states, they lack a representation of knowledge. **Logic** provides the framework for this representation.

1.1 Knowledge-Based Agents

A **Knowledge-Based Agent** maintains a representation of the world and uses logical reasoning to derive new information and make decisions. It operates on two main components:

Components of a Knowledge-Based System

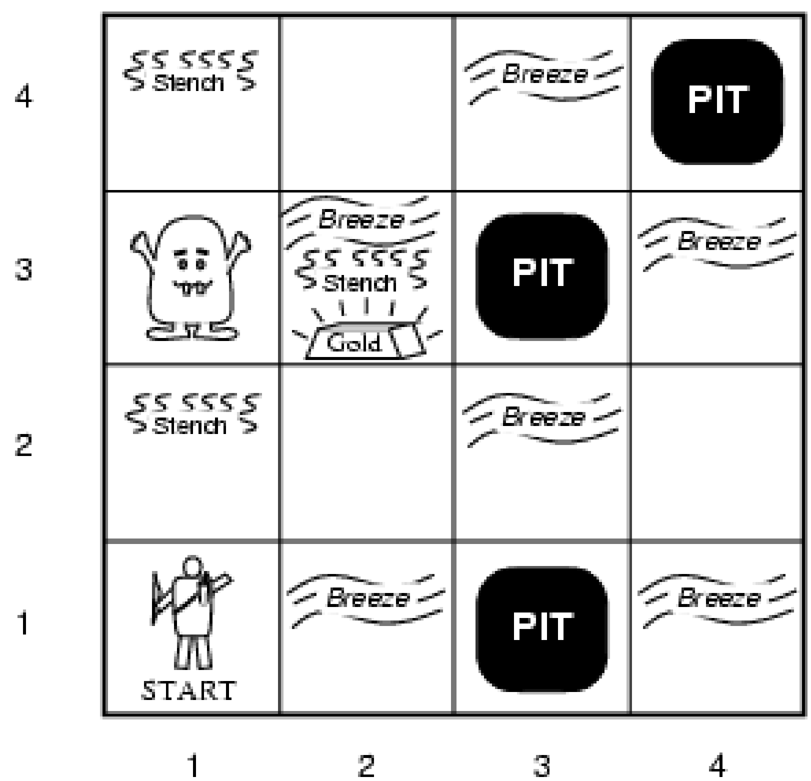
- **Knowledge Base (KB)**: A set of sentences in a formal language representing facts about the world. It follows a declarative approach (TELL the agent what it needs to know).
- **Inference Engine**: Domain-independent algorithms that derive new sentences (conclusions) from the KB.

The interaction loop of a KB-Agent involves:

1. **TELL**: The agent incorporates new percepts into the KB.
2. **ASK**: The agent queries the KB to decide on an action.
3. **TELL**: The agent records the chosen action and updates the time.

1.2 The Wumpus World

The Wumpus World is a standard environment used to illustrate logical reasoning in AI. It is a grid-based cave where an agent must find gold while avoiding pits and a monster (Wumpus).



1.2.1 PEAS Description

- **Performance**: $+1000$ for gold, -1000 for death, -1 per step, -10 for using the arrow.
- **Environment**:
  - Squares adjacent to the **Wumpus** smell (Stench).
  - Squares adjacent to a **Pit** are breezy (Breeze).
  - Gold glitters in its square.
- **Sensors**: $[Stench, Breeze, Glitter, Bump, Scream]$.
- **Actuators**: Turn Left/Right, Forward, Grab, Release, Shoot.

1.2.2 Reasoning Example

If the agent is in $[1, 1]$ and perceives no breeze and no stench, it knows $[1, 2]$ and $[2, 1]$ are safe (OK). If it moves to $[2, 1]$ and perceives a breeze, it infers a pit must be in $[2, 2]$ or $[3, 1]$. Logic allows the agent to combine observations over time to build a map of safe and dangerous areas.

1.3 Propositional Logic (PL)

Propositional logic is the simplest logic, where symbols represent whole propositions (facts) that can be true or false.

1.3.1 Syntax

Syntax defines the rules for constructing well-formed sentences. We use **Backus-Naur Form (BNF)**:

- **Atomic Sentences**: Single symbols (e.g., P, Q, $RoommateWet$).
- **Complex Sentences**: Constructed using logical connectives.
  - $\neg P$ (Not/Negation)
  - $P \land Q$ (And/Conjunction)
  - $P \lor Q$ (Or/Disjunction)
  - $P \Rightarrow Q$ (Implication/If-Then)
  - $P \Leftrightarrow Q$ (Biconditional/If and only if)

1.3.2 Semantics

Semantics defines the meaning of sentences, specifically their **Truth Value** relative to a specific world configuration (Interpretation).

Model

A **Model** is an interpretation (a specific setting of true/false values for all propositional symbols) in which a specific sentence or Knowledge Base is **True**.

Truth Tables The semantics are defined by truth tables. Key logical behaviors to remember:

- $P \land Q$ is true only if *both* are true.
- $P \lor Q$ is true if *at least one* is true (inclusive OR).
- $P \Rightarrow Q$ is true unless P is true and Q is false. (Note: $False \Rightarrow True$ is valid/True).

a	b	NOT(a)	a AND b	a OR b	a => b
false	false	true	false	false	true
false	true	true	false	true	true
true	false	false	false	true	false
true	true	false	true	true	true

1.3.3 Logical Properties

1. **Tautology**: A sentence that is true in *all* possible models (e.g., $P \vee \neg P$).
2. **Satisfiability**: A sentence is satisfiable if it is true in *at least one* model.
3. **Contradiction**: A sentence that is false in all models (e.g., $P \wedge \neg P$).
4. **Logical Equivalence**: Two sentences α and β are equivalent ($\alpha \equiv \beta$) if they have the same truth value in every model.

Important Equivalences (for simplification)

- **Double Negation**: $\neg(\neg A) \equiv A$
- **Contraposition**: $(A \Rightarrow B) \equiv (\neg B \Rightarrow \neg A)$
- **Implication Elimination**: $(A \Rightarrow B) \equiv (\neg A \vee B)$
- **De Morgan's Laws**:
  - $\neg(A \wedge B) \equiv (\neg A \vee \neg B)$
  - $\neg(A \vee B) \equiv (\neg A \wedge \neg B)$
- **Distributivity**: $A \wedge (B \vee C) \equiv (A \wedge B) \vee (A \wedge C)$

1.4 Inference and Entailment

The core goal of the Inference Engine is to determine if a sentence follows from the Knowledge Base.

Entailment ($KB \models \alpha$)

We say the Knowledge Base KB **entails** sentence α if and only if α is true in all models where KB is true.

1.4.1 Model Checking

A simple algorithm to check entailment is **Truth Table Enumeration**:

1. Enumerate all possible assignments of True/False to the symbols.
2. Check if the KB is true in that assignment.
3. If KB is true, check if α is also true.
4. If α is true in every model where KB is true, then $KB \models \alpha$.

Drawback: The time complexity is $O(2^n)$, making it inefficient for large numbers of variables.

1.4.2 Consistency and The Principle of Explosion

It is critical that a Knowledge Base is **Consistent**.

- If a KB contains a contradiction (e.g., P and $\neg P$), it is inconsistent.
- An inconsistent KB entails **everything**.

- *Example*: If "The roommate flies" and "The roommate does not fly" are both in the KB, we can prove "The Moon is made of cheese."
- This relies on the **Law of Non-Contradiction** (Aristotle): A and $\neg A$ cannot both be true.

1.5 Resolution and Proof Systems

To avoid enumerating truth tables, we use syntactic proof systems like **Resolution**. To use resolution, sentences must be in a specific form.

1.5.1 Conjunctive Normal Form (CNF)

Any sentence in propositional logic can be converted into CNF. A CNF formula is a conjunction (AND) of clauses, where each clause is a disjunction (OR) of literals.

$$(L_{1,1} \vee \cdots \vee L_{1,k}) \wedge (L_{2,1} \vee \dots) \wedge \dots$$

where a literal is a symbol (P) or its negation ($\neg P$).

Conversion Steps (Example):

1. Eliminate $\Leftrightarrow$ and $\Rightarrow$ using $(A \Rightarrow B) \equiv (\neg A \vee B)$.
2. Move $\neg$ inwards using De Morgan's Laws.
3. Distribute $\vee$ over $\wedge$.

1.5.2 The Resolution Rule

The resolution inference rule takes two clauses and produces a new one:

$$(P \vee A) \text{ and } (\neg P \vee B) \text{ derive } (A \vee B)$$

Here, P and $\neg P$ are complementary literals. They "cancel out."

Unit Resolution

A simplified version where one clause is a single literal:

$$(l_1 \vee \cdots \vee l_k) \text{ and } \neg l_i \implies (l_1 \vee \cdots \vee l_{i-1} \vee l_{i+1} \dots)$$

This is effectively **Modus Ponens** in CNF form.

1.5.3 Proof by Refutation (Resolution Algorithm)

To prove that $KB \models \alpha$, we use **Proof by Contradiction**:

1. Assume $\neg\alpha$ (the negation of what we want to prove).
2. Add $\neg\alpha$ to the Knowledge Base.
3. Convert the entire set of sentences to CNF.
4. Repeatedly apply the Resolution Rule to pairs of clauses containing complementary literals.
5. If you derive the **Empty Clause** (a contradiction, e.g., resolving P and $\neg P$), then the original assumption $\neg\alpha$ must be false, meaning α is true.

1.6 Horn Clauses

Resolution is complete (can prove anything that is true) but can be slow (exponential in worst case). **Horn Clauses** are a subset of PL that allows for more efficient inference.

- A Horn Clause is a disjunction of literals with **at most one positive literal**.

- *Example*: $\neg P \vee \neg Q \vee R$ is equivalent to $(P \wedge Q) \Rightarrow R$.
- Inference with Horn Clauses can be done in linear time using **Forward Chaining**.

1.7 Limitations of Propositional Logic

While powerful, PL has distinct limitations:

1. **Lack of Objects and Relations**: In PL, "Roommate carrying umbrella" is a single atomic symbol. The logic does not understand that "Roommate" is a person or "Umbrella" is an object.
2. **Verbose**: To say "All pits cause breezes," we must write a rule for every square: $B_{1,1} \Leftrightarrow (P_{1,2} \vee P_{2,1})$, $B_{1,2} \Leftrightarrow \ldots$
3. **Variable Explosion**: In the Wumpus world alone, a small grid generates 64 distinct symbols and hundreds of sentences.

These limitations lead to the development of **First-Order Logic** (not covered in this summary).