

Logic-Based Agents

Intelligent Systems II

Mário Antunes

2026

Universidade de Aveiro

Logic-Based Agents: Motivation and Agent Structure

Introduction to Logic-Based Agents

- **Intelligent Agent:** An entity that perceives its environment via sensors and acts upon it via actuators.
- Logic-based agents use a **Knowledge Base (KB)** to store facts/rules and apply **Inference** to decide actions.
- **Motivating Example:** Imagine a robot in a maze. Its sensors detect walls; its KB contains navigation rules.

Agent Function vs. Agent Program

- **Agent Function:** Abstract mapping:
 $f : \text{Percepts} \rightarrow \text{Actions}$
- **Agent Program:** Concrete code implementing this function—often as an **Inference Engine** in logic-based agents.
- **Example:** The agent's KB: "If you see a wall, turn left."
The inference engine deduces action.
Difference: Function defines what should happen; program determines how.

Paradigms: “What” vs. “How”

- **Imperative Programming:** (Python, Java, C++) — Sequence of instructions (“do this, then that”)
- **Declarative Programming:** (Prolog, SQL, Logic) — Describes properties/solutions (“it should be true that...”)
- **Analogy:** Imperative = giving step-by-step directions. Declarative = stating the destination.
- **Paradigm Shift:** In logic, you describe the problem; the engine finds the solution.

Example: Imperative vs Declarative

- **Imperative:**

```
def is_even(n):  
    return n % 2 == 0
```

- **Declarative (Prolog):**

```
even(0).  
even(N) :- N > 0, Prev is N - 2, even(Prev).
```

Insight: Imperative runs the steps; declarative lets the computer “search for how it’s true.”

The Language of Logic

Formal Syntax — BNF for First-Order Logic

Formula \rightarrow AtomicFormula
| Formula Connective Formula
| Variable Quantifier Formula
| \neg Formula
| (Formula)
AtomicFormula \rightarrow Predicate(Term, ...) | Term = Term
Term \rightarrow Function(Term, ...) | Constant | Variable
Connective $\rightarrow \Rightarrow$ | \wedge | \vee | \Leftrightarrow
Quantifier $\rightarrow \exists$ | \forall
Constant \rightarrow A | X1 | Paula | ...
Variable \rightarrow a | x | s | ...
Predicate \rightarrow Near | Color | ...
Function \rightarrow AgeOf | MotherOf | ...

Logic Notational Conventions

- **Naming:**
 - Predicate symbols, function symbols, and constants begin with uppercase (e.g., *Near*, *FatherOf*, *Pipe1*)
 - Variables begin with lowercase (e.g., *x*, *y*, *agent*)
- **Parentheses** are used to clarify the scope of terms and functions. - **Use of Quantifiers:**
 - Universal quantifiers (\forall) for “for all ...”
 - Existential quantifiers (\exists) for “there exists ...”

These conventions help ensure logic statements are clear and unambiguous, critical for both human and machine interpretation.

Truth Tables and Satisfiability (Propositional Logic) i

- **Truth Table Example:** Let A and B be propositional variables. The formula $A \wedge B$ is true only if both A and B are true.

A	B	$A \wedge B$
0	0	0
0	1	0
1	0	0
1	1	1

Truth Tables and Satisfiability (Propositional Logic) ii

- **Satisfiability:** A formula is **satisfiable** if there is *some* interpretation where it is true.
- **Tautology:** A formula is a **tautology** if it is true under *all* interpretations.
- **Model:** An assignment that makes a formula true is a *model* for that formula.
- **Common Mistakes:** Confusing $A \wedge B$ (both must be true) with $A \implies B$ (B is true if A is true).

Example: Interpretation and Model in First-Order Logic i

- **Domain (Universe):** $\{A, B, C, \text{Floor}\}$
- **Constants:** A, B, C, Floor
- **Predicate:** $\text{OnTop}(x, y)$ (interpreted as: x is on top of y)
- **Interpretation:**
 - $\text{OnTop}(B, A)$ is true
 - $\text{OnTop}(A, C)$ is true
 - $\text{OnTop}(C, \text{Floor})$ is true
 - $\text{OnTop}(x, y)$ is false for all other (x, y)

Example: Interpretation and Model in First-Order Logic ii

- **Model Table:**

x	y	$OnTop(x, y)$
B	A	true
A	C	true
C	Floor	true
...	...	false

- This interpretation/model gives truth values to all ground atoms built from the domain and predicates.

Common Mistakes in Logic and Quantifiers i

- **Quantifier order:**

- $\forall x \exists y P(x, y)$ (For each x , there is a y) \neq
 $\exists y \forall x P(x, y)$ (There is a y for all x)

- **Mixing connectives:**

- $\forall x (Studies(x, Oxford) \implies Smart(x))$ ("All Oxford students are smart.")
- Incorrect: $\forall x (Studies(x, Oxford) \wedge Smart(x))$
(means all people are Oxford students and all are smart).

- **Scope issues:**

- $\exists x Likes(x, y)$ —Who is y quantified over? Unclear if y is free or bound.

Common Mistakes in Logic and Quantifiers ii

- **Negation with quantifiers:**

- $\neg \forall x P(x) \equiv \exists x \neg P(x)$

- **Ambiguity:**

- Always check whether your statement matches the intended natural language meaning!

Conclusion: Always translate English to logic step by step and check it with an interpretation/model example.

Knowledge-Based Agent Loop

1. **RECEIVE:** Agent gets percept.
2. **TRANSLATE:** Converts percept to logic sentence.
3. **TELL:** Updates KB.
4. **ASK:** Queries KB for best action.
5. **RECORD:** Logs action.
6. **EXECUTE:** Performs action.

Example: Robot detects wall at (5,5) \rightarrow $Wall(5, 5)$
 \rightarrow KB \rightarrow "Turn left" action.

Propositional and First-Order Logic

Propositional Logic: The Foundation

- **Propositional Logic:** Deals with whole facts (atomic symbols)
 - Symbols: P, Q, R
 - Connectives: \neg (not), \wedge (and), \vee , \implies , \iff
- **Limitation:** Can't easily express "All students are here."
Requires separate symbols for each student.

First-Order Logic (FOL): Greater Expressiveness

- Introduces **Objects, Relations (Predicates), Functions**
 - **Objects:** John, Pipe1
 - **Predicates:** *Near(Bird, Pipe)*
 - **Functions:** *GapOf(Pipe1)*
- Enables powerful generalizations: "For all birds, they can fly (unless exceptions)."

Quantifiers: Universal and Existential

- **Universal (\forall):** "For all"
 - $\forall x \text{ Student}(x) \implies \text{Human}(x)$
- **Existential (\exists):** "There exists"
 - $\exists x \text{ Crown}(x) \wedge \text{OnHead}(x, \text{John})$
- **Analogy:** Universal = sweeping statement; Existential = finding at least one.

- **Term:** Constant, variable, or function (e.g., *Peter*, *x*, *BrotherOf(John)*)
- **Atomic Sentence:** Predicate applied to terms (e.g., *Sibling(Peter, BrotherOf(John))*)
- **Complex Sentence:** Combines atomic ones using connectives.

Reasoning and Inference

Inference Rules: Modus Ponens

- **Illustration:**

- Example chain: "If it is raining, then the ground is wet"
(*Raining* \implies *Wet*)
- If we know *Raining* is true, we can deduce *Wet* is true
- This illustrates how agents chain knowledge: $p \implies q$
and p , therefore q .

- **Modus Ponens Formalism:**

- $p \implies q, p \vdash q$
- This is the backbone of inference, used to reason from facts and rules and build longer chains of reasoning.

- **Example in Agent Context:**

- KB: "If there is a wall ahead, turn left."
- Percept: "There is a wall ahead."
- By Modus Ponens: Infer: "Turn left."

- **Unification:** Makes logical expressions match via substitution.
 - Example: $At(Agent, x)$ and $At(Agent, Kitchen)$ unify as $\{x/Kitchen\}$
- **Analogy:** Pattern-matching puzzle pieces to fit the current situation.

Worked Examples: Logic Agents in Action

Worked Example: Wumpus World as a Logic Agent i

- **Environment:** A 4×4 grid world with pits, a Wumpus (monster), and gold. Agent perceives breeze (adjacent to pits), stench (adjacent to Wumpus), and glitter (in same square as gold).
- **Atoms/Predicates:**
 - $Pit(x, y)$: pit at cell (x, y)
 - $Wumpus(x, y)$: wumpus at cell (x, y)
 - $Breeze(x, y)$: breeze in cell (x, y)
 - $Stench(x, y)$: stench in cell (x, y)
 - $Gold(x, y)$: gold in cell (x, y)
 - $Safe(x, y)$: cell (x, y) is safe

Worked Example: Wumpus World as a Logic Agent ii

- **Rules (Knowledge Base):**

- $Breeze(x, y) \iff Pit(x + 1, y) \vee Pit(x - 1, y) \vee Pit(x, y + 1) \vee Pit(x, y - 1)$
- $Stench(x, y) \iff Wumpus(x + 1, y) \vee Wumpus(x - 1, y) \vee Wumpus(x, y + 1) \vee Wumpus(x, y - 1)$
- $Safe(x, y) \implies \neg Pit(x, y) \wedge \neg Wumpus(x, y)$
- $\neg Breeze(1, 1) \implies \neg Pit(1, 2) \wedge \neg Pit(2, 1)$

- **Inference:**

- From $\neg Breeze(1, 1)$, agent deduces adjacent cells have no pits.
- From observed *Breeze/Stench*, agent can deduce (using resolution!) where pits/wumpus are/aren't.

- **Actions:**

- Move to adjacent safe cell. Grab gold if in same cell.
Shoot arrow at likely Wumpus location.

Worked Example: Missionaries and Cannibals i

- **Problem:** Three missionaries and three cannibals must cross a river in a boat that holds at most two. Cannibals can never outnumber missionaries on either side.
- **State Representation (FOL or Datalog-style):**
 - $State(MLeft, CLeft, BoatPos)$: $MLeft$ missionaries, $CLeft$ cannibals, boat is $BoatPos$ (left/right).
- **Initial State:** $State(3, 3, Left)$
- **Goal State:** $State(0, 0, Right)$

Worked Example: Missionaries and Cannibals ii

- **Transition Rules:**

- Valid transition if $1 \leq \#people \leq 2$ in boat, and never $C > M$ (if $M > 0$) on either side.
- Example action: $Move(1M, 1C)$:
 $MLeft' = MLeft - 1, CLeft' = CLeft - 1$

- **Safety Predicate:**

- $Safe(M, C) : \neg(M = 0) \vee (M \geq C)$

- **Action Rule Example:**

- $State(M, C, Left), Safe(M - 1, C), Safe(3 - M + 1, 3 - C) \implies State(M - 1, C, Right)$

- **Search:** Use logic inference to enumerate safe next states and avoid forbidden states.

The Mechanics of Unification: Rules

- Constants must match exactly.
- Variables bind to constants or other variables.
- Functions must have matching names and arguments.
- Enables flexible rule application: "If it fits, apply."

Resolution: Finding Contradictions

- **Resolution:** Combines clauses to infer new knowledge or detect contradictions.
 - Core in logical reasoning and automated theorem proving.
 - If KB and query cannot both be true, contradiction found; query is false.

Proof by Contradiction (Refutation)

- **Technique:** Add $\neg G$ (negation of goal) to KB, try to derive a contradiction.
- **Example:** Asking Prolog a question that has no solution returns “No”—a contradiction wasn’t possible.

Knowledge Representation and Change

- **Semantic Nets:** Graphs where nodes = objects/concepts, edges = relationships.
 - Enables inheritance (e.g., Mammal \rightarrow Cat \rightarrow "has fur")
- **Analogy:** Family tree for knowledge.

Description Logics (DL)

- Restrictive but efficient subset of FOL.
- Used to define **Concepts** (unary predicates), **Roles** (binary predicates).
- Backbone of ontologies and the Semantic Web.

- **Situation:** Sequence of actions performed.
- **Fluents:** Properties that can change (e.g., $At(Agent, Location, S)$)
- **Result Function:** $Result(Action, Situation)$ yields new situation after action.

The Frame Problem: What Doesn't Change? i

- **Definition:** The frame problem is the challenge of specifying, in logical terms, which facts about the world remain unchanged after an action—or, equivalently, how to avoid having to write a (potentially huge) number of axioms spelling out all the things that aren't affected by each possible action.
- **Illustrative example:** Suppose an agent moves a cup on a table. We must specify that the positions of the walls, the color of the cup, the temperature of the room, etc., all stay the same—unless stated otherwise. For every action, countless facts *don't* change, and listing all of them explicitly is tedious and error-prone.

The Frame Problem: What Doesn't Change? ii

- **Successor-State Axioms** simplify this by compactly specifying what changes and what doesn't, reducing the need for redundant statements.
- **Takeaway:** The frame problem demonstrates the complexity of representing change efficiently in logic.
- When agent moves book, does wall color change?
No—but logic must state that explicitly!
- **Successor-State Axioms:** Specify what is preserved vs. changed.
 - *Fluent is true in next state \iff [Action makes it true \vee (It was true \wedge Action doesn't change it)]*

Rules, Search, and Applications

Rule-Based Systems: Forward Chaining

- **Data-driven:** Starts with facts, applies rules to discover new facts.
- Useful for real-time sensing/monitoring.
- **Example:** Expert system diagnosing disease from symptoms.

Rule-Based Systems: Backward Chaining

- **Goal-driven:** Starts with desired outcome, works backward to see if rules + facts can support it.
- **Prolog:** Uses backward chaining for queries.

- **Horn Clause:** Disjunction with at most one positive literal.
 - $\neg P_1 \vee \dots \vee \neg P_n \vee Q$
 - Prolog notation: $Q : \neg P_1, P_2, \dots, P_n$
- Restriction enables efficient resolution.

- Agents may face many rules—must search for applicable ones.
- **Prolog:** Uses **Depth-First Search (DFS)** with **Backtracking**.
 - Follows one path until failure, then rewinds and tries alternatives.

- **Expert System:** Mimics human expert (doctor, engineer, etc.)
 - Separation between **Inference Engine** (reasoning) and **Knowledge Base** (domain facts).
- Changing facts doesn't require rewriting reasoning software.

- Agents communicate through logical statements.
 - **Speech Acts:** Telling, asking, requesting.
- **Definite Clause Grammars (DCG):** Parse natural language into logical forms.
 - Example: "Pick up the block"
→ *Action(Pickup, Block1)*

Logical Agent Summary

- Logic-based agents:
 - **Symbolic:** Use formal language.
 - **Transparent:** Reasoning is inspectable.
 - **Rigorous:** Provides mathematical guarantees.
 - Foundations for advanced AI: planning, ontologies, inductive logic, expert systems.