Beyond Reflexes: Building Agents That Reason

Simple problem-solving agents are powerful but brittle. They know procedures, not general facts about the world. Examples:

* **An 8-puzzle agent doesn't know that two tiles cannot occupy the same space.**
* **A route-finding agent doesn't know a road cannot be a negative number of kilometres long.**

The Solution:

We need **Knowledge-Based Agents**. These agents possess an internal representation of the world—a **Knowledge Base (KB)**—and use **inference** to derive new knowledge and make decisions.

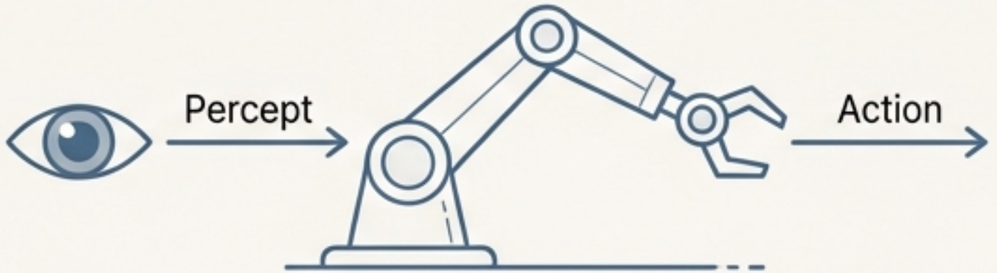
Core Components of a KB-Agent:

- **Knowledge Base (KB)**: A set of sentences expressed in a formal language.
- **Inference Engine**: A mechanism to derive new sentences from old ones.

Key Operations:

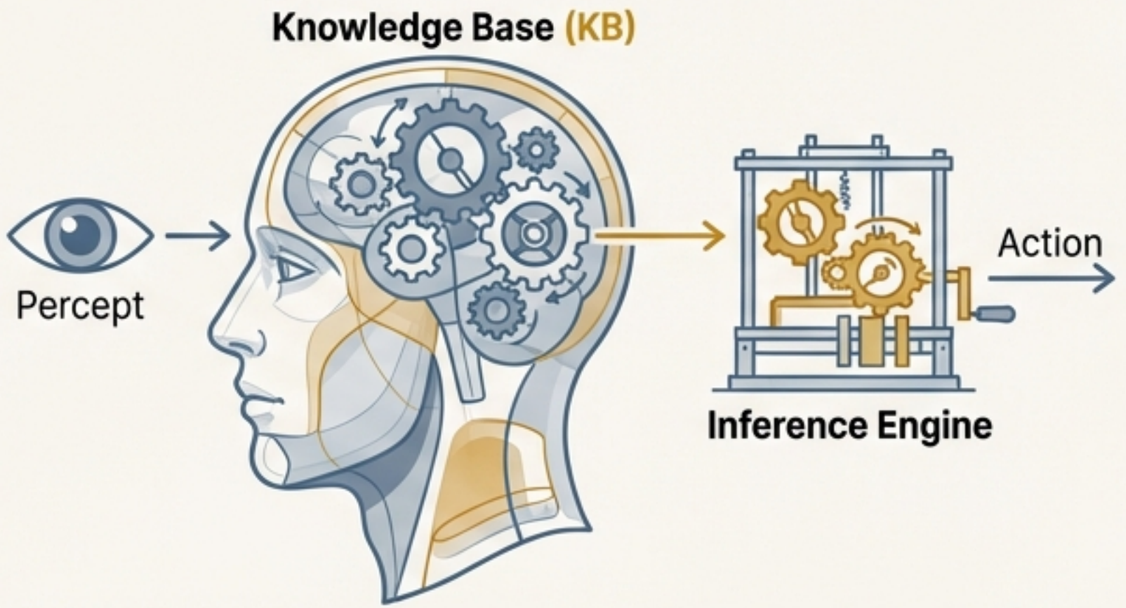
- **TELL**: Add new sentences (percepts, facts) to the KB.
- **ASK**: Query the KB to determine the best action. The answer must *follow logically* from what the KB has been told.

Reactive Agent



Knowledge-Based Agent

Percept → Update KB → Reason → Action



The Agent's Inner World: The Knowledge Base Loop

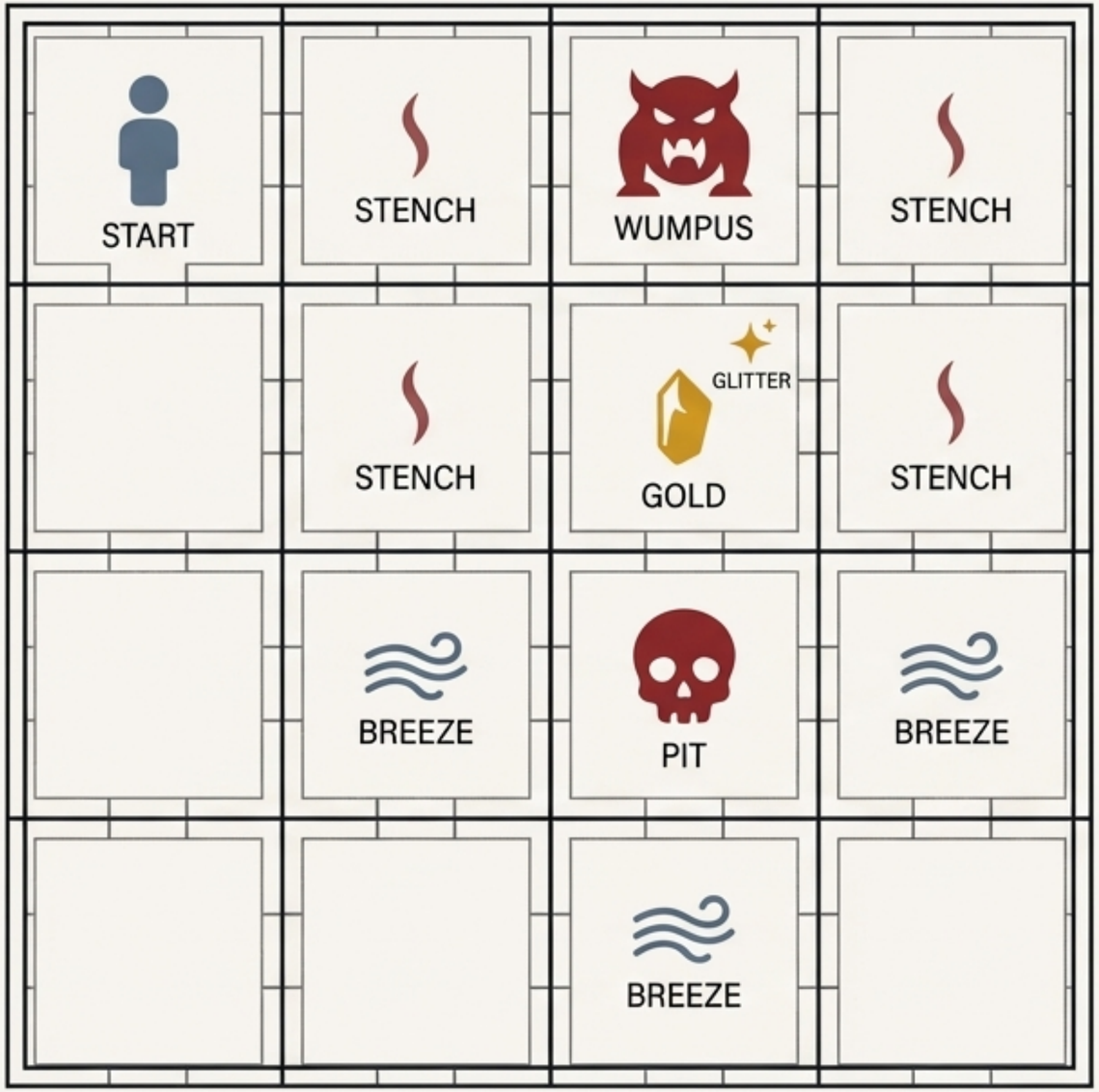
```
function KB-AGENT(percept) returns an action
   persistent: KB, a knowledge base
               t, a counter, initially 0

   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
```

Explanation of the Loop:

1. **PERCEIVE**: The agent perceives its environment.
2. **TELL**: It translates the percept into a formal sentence and adds it to its KB.
3. **ASK**: It queries the KB for the best action to take. This is where inference happens.
4. **ACT & TELL**: It performs the chosen action and informs the KB of its choice.

Key Takeaway: This architecture separates knowledge from the reasoning mechanism. We can update the agent's knowledge (the **declarative** approach) without reprogramming its core logic (the **procedural** approach).

The Quest Begins: Navigating the Wumpus World



The Rules of the World

- **Goal**: Find the **Gold** (+1000 points) and return to [1,1].
- **Hazards**:
  - **Wumpus**: A beast that kills any agent entering its room. A **Stench** can be perceived in adjacent squares.
  - **Pits**: Bottomless pits that trap any agent entering. A **Breeze** can be perceived in adjacent squares.
- **Reward**: The **Glitter** is only perceived in the same square as the Gold.

The Core Challenge

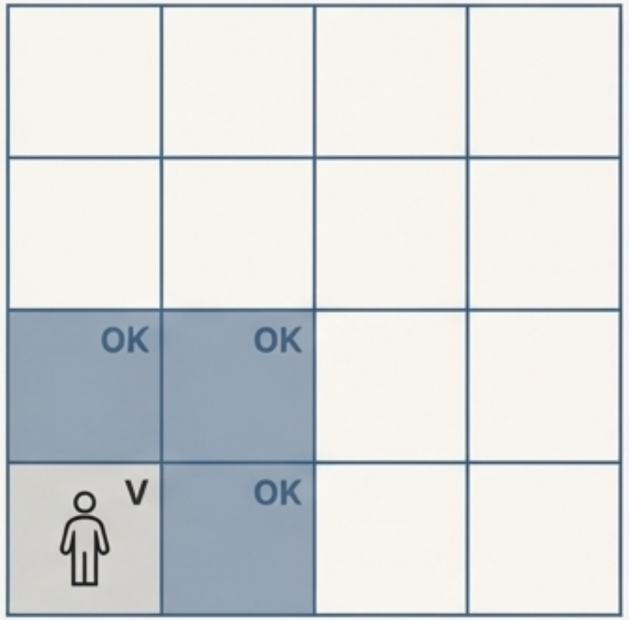
The agent is initially ignorant of the world's layout. It must use its percepts to reason about the locations of pits and the wumpus to explore safely.

PEAS Description

Performance +1000 for gold, -1000 for death, -1 per step, -10 for arrow	**Environment** 4x4 grid, randomly placed Wumpus and Gold
Actuators Turn Left, Turn Right, Forward, Grab, Shoot	**Sensors** [Stench, Breeze, Glitter, Bump, Scream]

First Steps into the Darkness

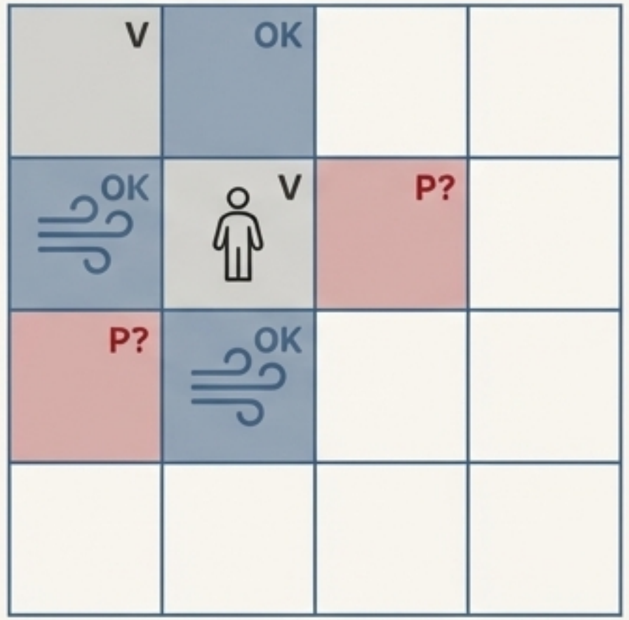
Grid 1: The Start [1,1]



Percept in [1,1]: [None, None, None, None, None].

Inference: No Stench or Breeze means adjacent squares [1,2] and [2,1] are safe.

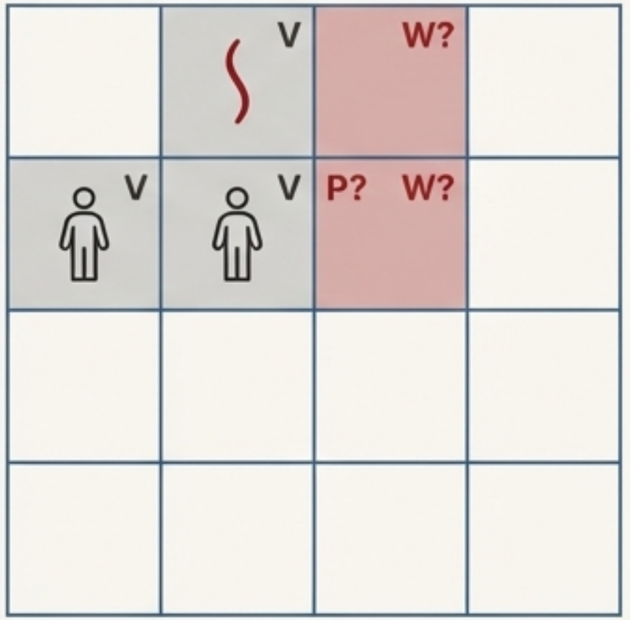
Grid 2: A Move to [2,1]



Percept in [2,1]: [None, **Breeze**, None, None, None].

Inference: A Breeze implies a pit in [1,1], [2,2], or [3,1]. We know $\neg Pit_{1,1}$ from visiting. Therefore, the agent concludes $Pit_{2,2}$ V $Pit_{3,1}$.

Grid 3: Retreat and Rethink



Percept in [1,2]: [**Stench**, None, None, None, None].

Inference: A Stench implies the Wumpus is in an adjacent square. The agent deduces the Wumpus is in [1,3] or [2,2].

The Language of Truth: Formal Logic

To reason rigorously, our agent needs a **formal language** with two key properties:

- **Syntax**: Defines the rules for forming well-formed sentences.
- **Semantics**: Defines the meaning of sentences by connecting them to the truth in a possible world (a 'model'). Every sentence is either **true** or **false** in a given model.

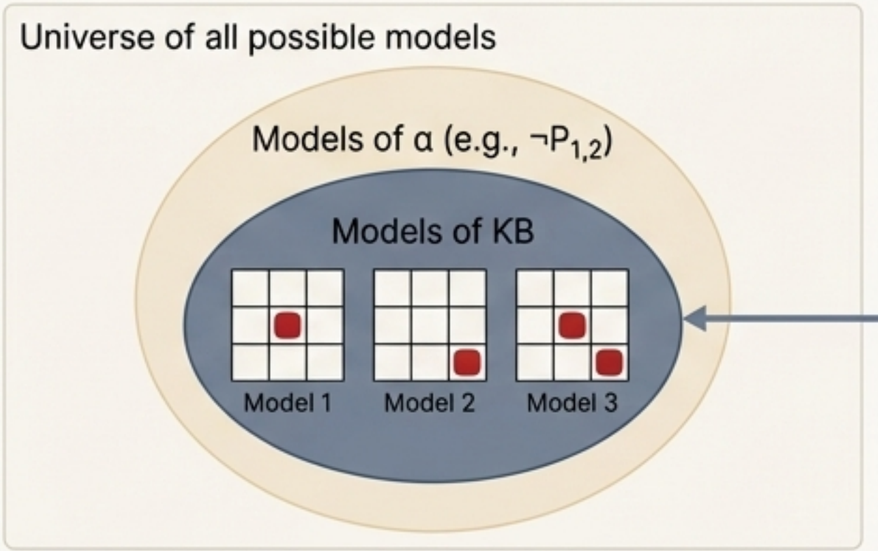
The Crucial Con: The Crucial Concept: Logical Entailment

$$KB \models \alpha$$

The Knowledge Base KB entails the sentence α.

Definition: α is entailed by KB if and only if, in every model where KB is true, α is also true.

In Simple Terms: Entailment means α is a guaranteed conclusion from KB. The inference process should not 'make things up'.



The set of models where KB is true is a subset of the models where α is true.
$M(KB) \subseteq M(\alpha)$.
Since all three valid KB models are inside the $\neg P_{1,2}$ circle, the agent can conclude $\neg P_{1,2}$ is true.

Forging a Tool: An Introduction to Propositional Logic

Syntax: The Building Blocks

- **Atomic Sentences**: A single proposition symbol (e.g., $P_{1,2}$, $W_{1,3}$, $B_{2,1}$). Each stands for a proposition that can be true or false.
- **Complex Sentences**: Built by combining simpler sentences with five logical connectives.

The Five Connectives (Operators):

- ¬ (Not): Negation
- ∧ (And): Conjunction
- ∨ (Or): Disjunction
- ⇒ (Implies): Implication
- ⇔ (If and only if): Biconditional

P	Q	¬P	P ∧ Q	P ∨ Q	P ⇒ Q	P ⇔ 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

Semantics: The Rules of Truth

A **model** in propositional logic is simply an assignment of true/false to every proposition symbol.

Operator Precedence (highest to lowest): ¬, ∧, ∨, ⇒, ⇔

Translating the World into Logic

1	**Define what the symbols mean.**	$P_{x,y}$: There is a pit in [x,y]. $W_{x,y}$: There is a wumpus in [x,y]. $B_{x,y}$: The agent perceives a breeze in [x,y]. $S_{x,y}$: The agent perceives a stench in [x,y].
2	**State general facts about the environment.**	Rule 1: There is no pit in the starting square. R_1: $\neg P_{1,1}$ Rule 2: The starting square. R_2: $B_{1,1} \Leftrightarrow (P_{1,2} \vee P_{2,1})$ R_3: $B_{2,1} \Leftrightarrow (P_{1,1} \vee P_{2,2} \vee P_{3,1})$
3	**Add what the agent perceives perceives over time.**	The agent is in [1,1] and perceives nothing. Then it moves to [2,1] and perceives a breeze. R_4: $\neg B_{1,1}$ R_5: $B_{2,1}$

The Agent's Knowledge Base (KB):

The agent's total knowledge is the conjunction of all these sentences:

$$KB = R_1 \wedge R_2 \wedge R_3 \wedge R_4 \wedge R_5$$

Can the agent now prove that square [1,2] is safe? In formal terms: Does $KB \models \neg P_{1,2}$?

The Brute-Force Method: Inference by Model Checking

The Algorithm

1. Enumerate all possible models (all possible true/false assignments to proposition symbols).
2. Check that in every model where KB is true, α is also true.

Application to the Wumpus World

Our KB involves 7 proposition symbols ($B_{1,1}$, $B_{2,1}$, $P_{1,1}$, $P_{1,2}$, $P_{2,1}$, $P_{2,2}$, $P_{3,1}$). This means there are $2^7 = 128$ possible models to check.

...	$P_{1,2}$	$P_{2,2}$	...	**KB**
...	false	false	...	false
...	⋮	⋮	...	⋮
...	false	false	...	**true**
...	false	true	...	**true**
...	false	true	...	**true**
...	⋮	⋮	...	⋮

The Result

Is $\neg P_{1,2}$ entailed?

In all three models where KB is true, $P_{1,2}$ is false. Therefore, $\neg P_{1,2}$ is true.

Conclusion: YES, KB $\models \neg P_{1,2}$. Square [1,2] is safe.

Is $\neg P_{2,2}$ entailed?

In the models where KB is true, $P_{2,2}$ is sometimes true and sometimes false.

Conclusion: NO, the agent cannot be sure about a pit in [2,2].

Limitation: Model checking is sound and complete, but it is computationally intractable for large KBs.

A More Elegant Weapon: Inference by Theorem Proving

The Concept: Instead of enumerating models, we can apply **inference rules** directly to the sentences in the KB to generate a **proof**—a chain of logical steps that leads to the desired conclusion. This can be far more efficient.

Key Tool 1: Logical Equivalences

Two sentences are logically equivalent ($\alpha \equiv \beta$) if they are true in the same set of models. A table of crucial equivalences is presented below in a clean, well-spaced format.

Implication Elimination	$(\alpha \Rightarrow \beta) \equiv (\neg\alpha \vee \beta)$
Contraposition	$(\alpha \Rightarrow \beta) \equiv (\neg\beta \Rightarrow \neg\alpha)$
De Morgan's Laws	$\neg(\alpha \wedge \beta) \equiv (\neg\alpha \vee \neg\beta)$
De Morgan's Laws	$\neg(\alpha \vee \beta) \equiv (\neg\alpha \wedge \neg\beta)$

Key Tool 2: Inference Rules

The following rules allow us to derive new, true sentences.

Modus Ponens (The mode that affirms)

The most famous rule. From an implication and its premise, you can infer the conclusion.

$$\frac{\alpha \Rightarrow \beta, \quad \alpha}{\beta}$$

And-Elimination

From a conjunction, you can infer any of the conjuncts.

$$\frac{\alpha \wedge \beta}{\alpha}$$

The Proof: Deducing Safety in the Wumpus World

Goal: Prove $\neg P_{1,2}$ from the KB.

Initial Knowledge

We start with these sentences from our Knowledge Base:

- R_2: $B_{1,1} \Leftrightarrow (P_{1,2} \vee P_{2,1})$
- R_4: $\neg B_{1,1}$

The Derivation (A Step-by-Step Proof)

1. Start with R_2: $B_{1,1} \Leftrightarrow (P_{1,2} \vee P_{2,1})$
2. Biconditional Elimination on R_2: This is equivalent to two implications.
   R_6: $(B_{1,1} \Rightarrow (P_{1,2} \vee P_{2,1})) \wedge ((P_{1,2} \vee P_{2,1}) \Rightarrow B_{1,1})$
3. And-Elimination on R_6: We only need the second part. R_7: $(P_{1,2} \vee P_{2,1}) \Rightarrow B_{1,1}$
4. Logical Equivalence (Contrapositive) on R_7: This flips the implication. R_8: $\neg B_{1,1} \Rightarrow \neg(P_{1,2} \vee P_{2,1})$
5. Modus Ponens with R_8 and R_4 ($\neg B_{1,1}$): We have the premise, so we infer the conclusion. R_9: $\neg(P_{1,2} \vee P_{2,1})$
6. De Morgan's Rule on R_9: This distributes the negation. R_{10}: $\neg P_{1,2} \wedge \neg P_{2,1}$

Conclusion

$\neg P_{1,2} \wedge \neg P_{2,1}$

We have successfully proven that neither [1,2] nor [2,1] contains a pit. The agent can confidently move to [1,2].

The Universal Tool: Inference by Resolution

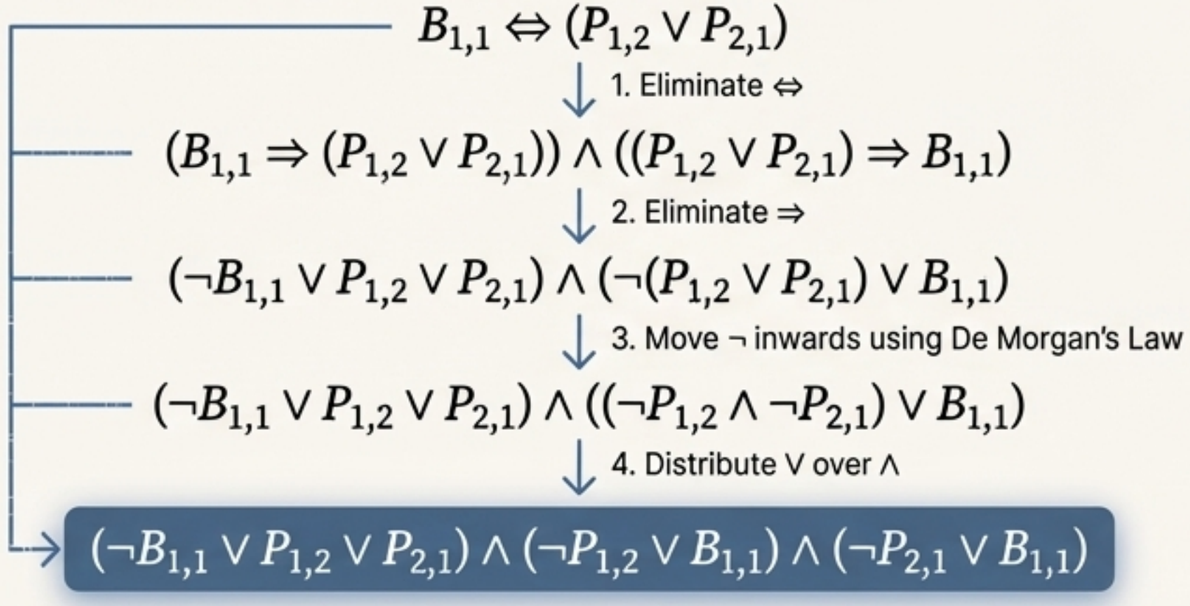
The Power of Resolution

A single rule that, when combined with a complete search algorithm, provides a complete inference procedure for all of propositional logic. It works by proving a contradiction.

The Prerequisite: Conjunctive Normal Form (CNF)

Resolution only works on sentences that are a conjunction of clauses, where each clause is a disjunction of literals. Any propositional logic sentence can be converted to an equivalent CNF sentence.

Visual: Example Conversion to CNF



The Resolution Rule

Given two clauses containing complementary literals (l_i and $\neg l_i$), a new clause can be generated containing all literals except the complementary pair.

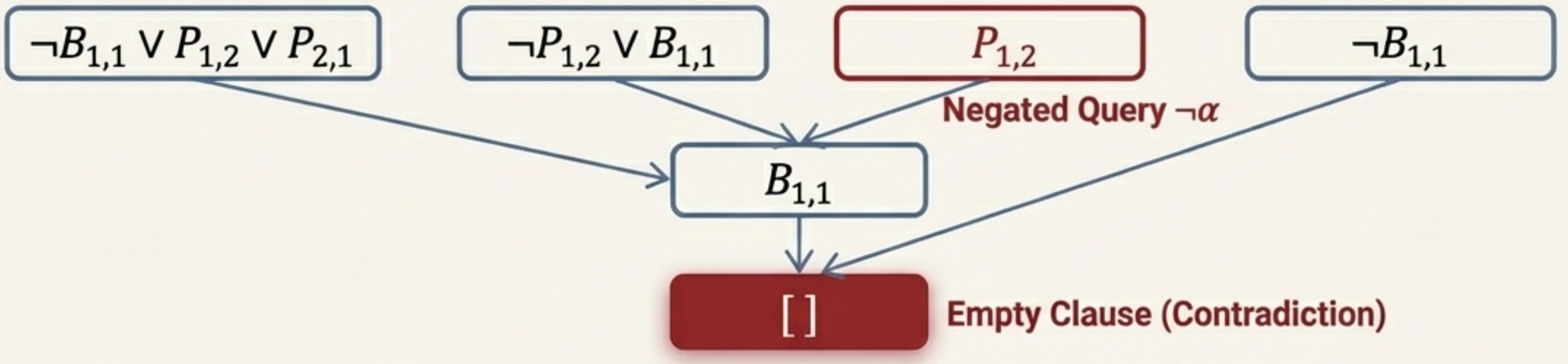
$$\frac{l_1 \vee \ldots \vee l_k, \qquad m_1 \vee \ldots \vee m_n \qquad l_i = \neg m_j}{(l_1 \vee \ldots \vee l_{i-1} \vee l_{i+1} \vee \ldots \vee l_k \vee m_1 \vee \ldots \vee m_{j-1} \vee \ldots)}$$

Proof by Contradiction: The Resolution Algorithm

The Algorithm PL-RESOLUTION

1. Convert the knowledge base KB and the *negation* of the query $\neg\alpha$ into CNF.
2. Combine all resulting clauses into a single set.
3. Repeatedly apply the resolution rule to pairs of clauses that contain complementary literals.
4. Add the resulting resolvent clause to the set.
5. If the **empty clause** (representing a contradiction) is derived, the original query α is entailed.

Visual: The Resolution Graph



Conclusion: Since we derived a contradiction (the empty clause) from $KB \wedge P_{1,2}$, the original assumption must be false. Therefore, the query $\neg P_{1,2}$ must be true. The proof is complete.

Practical Reasoning: Horn Clauses and Chaining

A Restricted but Common Form

Many real-world KBs use a special structure that allows for faster inference.

- **Horn Clause**: A disjunction of literals with *at most one* positive literal.
  - e.g., $\neg P_1 \wedge \neg P_2 \wedge \ldots \wedge \neg P_n \vee H$
- **Definite Clause**: A Horn clause with *exactly one* positive literal.

Why They Matter

1. **Intuitive Form**: Every definite clause can be written as an implication: $(P_1 \wedge P_2 \wedge \ldots \wedge P_n) \Rightarrow H$. The conjunction of positive literals (the body) implies a single positive literal (the head). This is easy for humans to understand.
2. **Efficient Inference**: Entailment for Horn clause KBs can be decided in time linear to the size of the KB.

Two Key Algorithms for Definite Clauses

Forward Chaining (Data-Driven)

Starts with known facts. When all premises of a rule are satisfied, its conclusion is added as a new fact. Repeats until the goal is found or no new facts can be added.

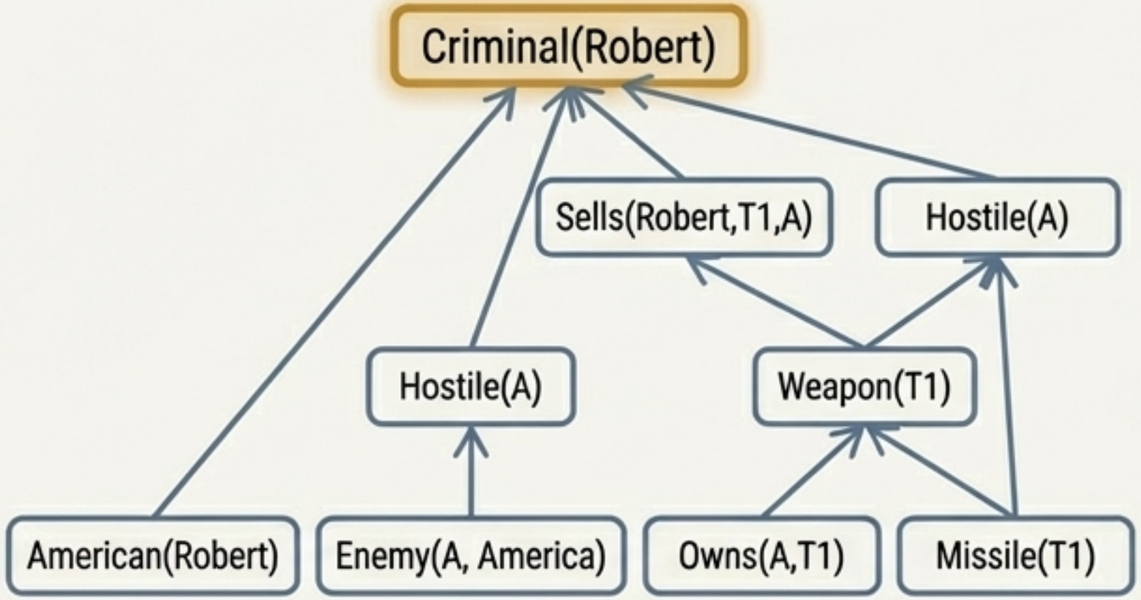
Backward Chaining (Goal-Driven)

Starts with the query. To prove the query, it finds rules that conclude it and then tries to prove their premises (sub-goals). Works backwards until it reaches known facts.

Two Paths to a Conclusion: Forward vs. Backward Chaining

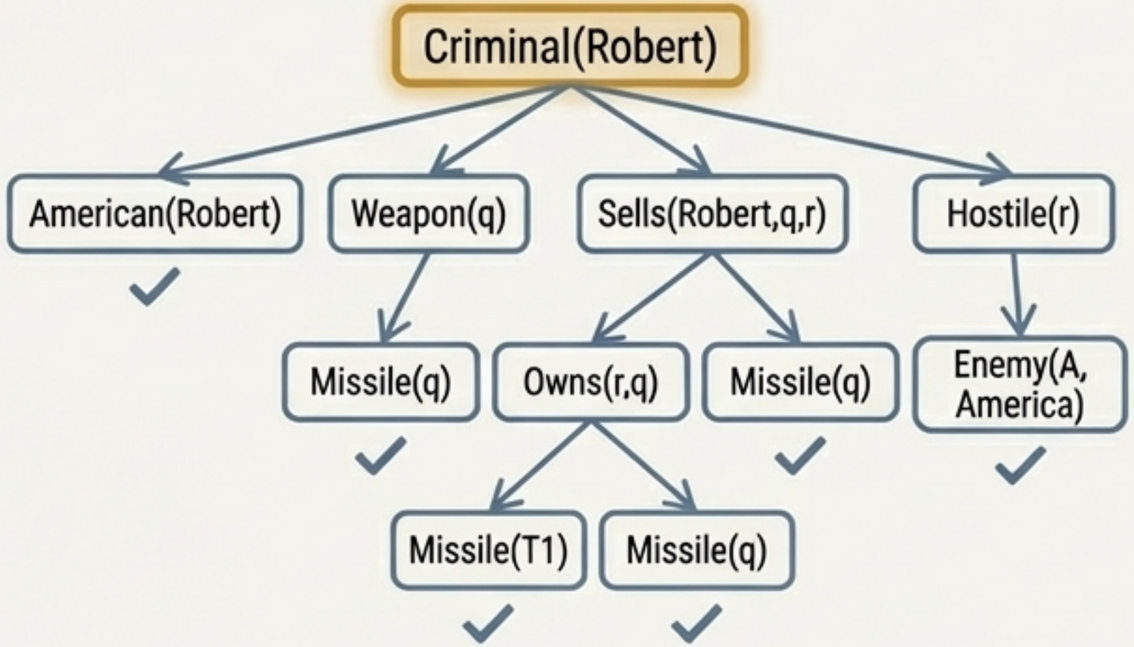
Forward Chaining: Data-Driven Reasoning

"What new conclusions can I draw from this new piece of information?"



Backward Chaining: Goal-Driven Reasoning

"To prove this goal, what sub-goals must I prove first?"



When to Use Which?

- **Forward Chaining:** Good when new facts arrive and you want to see all consequences. Used in expert systems and production rules.
- **Backward Chaining:** Good for answering specific queries, as it only touches relevant facts. Used in proof assistants and diagnostics.

The Limits of a Propositional World

Success of the Quest

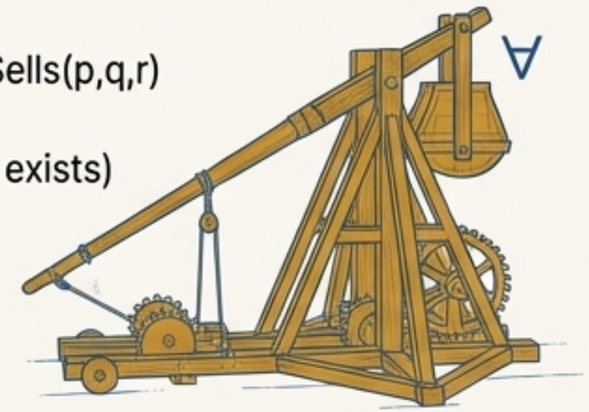
Using Propositional Logic, our agent successfully navigated its world by:

- Representing world knowledge (R_1: $\neg P_{1,1}$).
- Incorporating percepts (R_5: $B_{2,1}$).
- Deducing hidden properties of the world ($KB \models \neg P_{1,2}$).

The Next Horizon: First-Order Logic (FOL)

To overcome these limits, our agent needs a more powerful language. FOL introduces:

- Objects: John, A, T1
- Relations (Predicates): King(x), Sells(p,q,r)
- Functions: FatherOf(John)
- Quantifiers: ∀ (For all), ∃ (There exists)



The journey from simple reflexes to propositional reasoning is the first great step towards true artificial intelligence. The next step is to reason about a world of objects and their relationships.