KNOWLEDGE REPRESENTATION AND REASONING

BY VAISHNAVEE RATHOD

AI PROBLEM AREAS/TASK

1stGeneration of AI : Formal cognitive Tasks

–Game

- Tic-Tac-Toe
- Chess
- Checkers
- Go

–Mathematics

- Logic
- Geometry
- Calculus
- Proving properties of programs \

AI PROBLEM AREAS/TASK

2ndGeneration : Expert Tasks

–Knowledge Represenation

–Engineering

- Design
- fault finding
- Manufacturing planning

–Medical

- Diagnosis
- Medical Image Analysis

–Finanical

- Stock market prdeictions

AI PROBLEM AREAS/TASK

3rd Generation of AI : Perceptual Tasks

–Perception

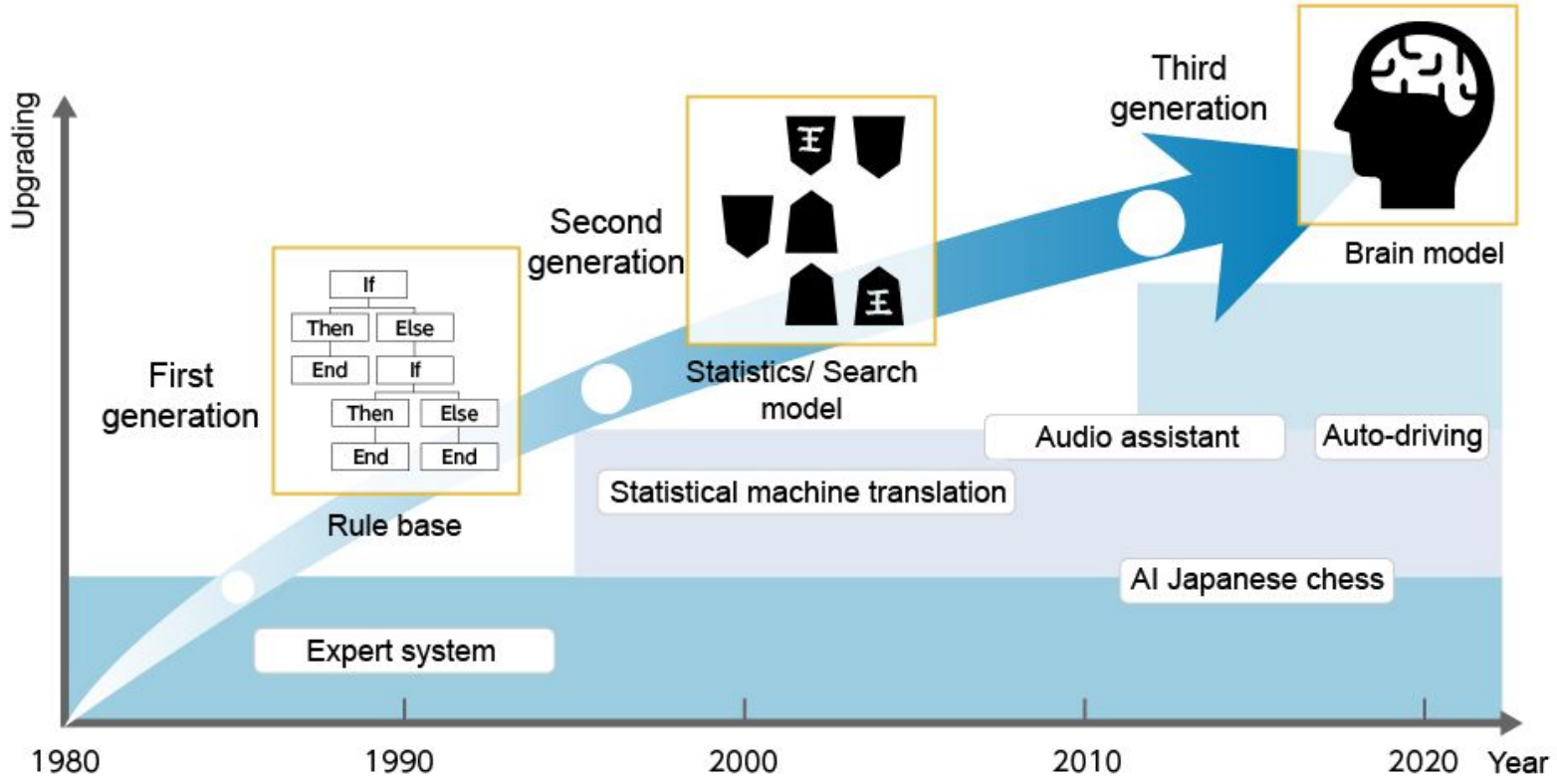
- Vision
- Speech

–Natural Language

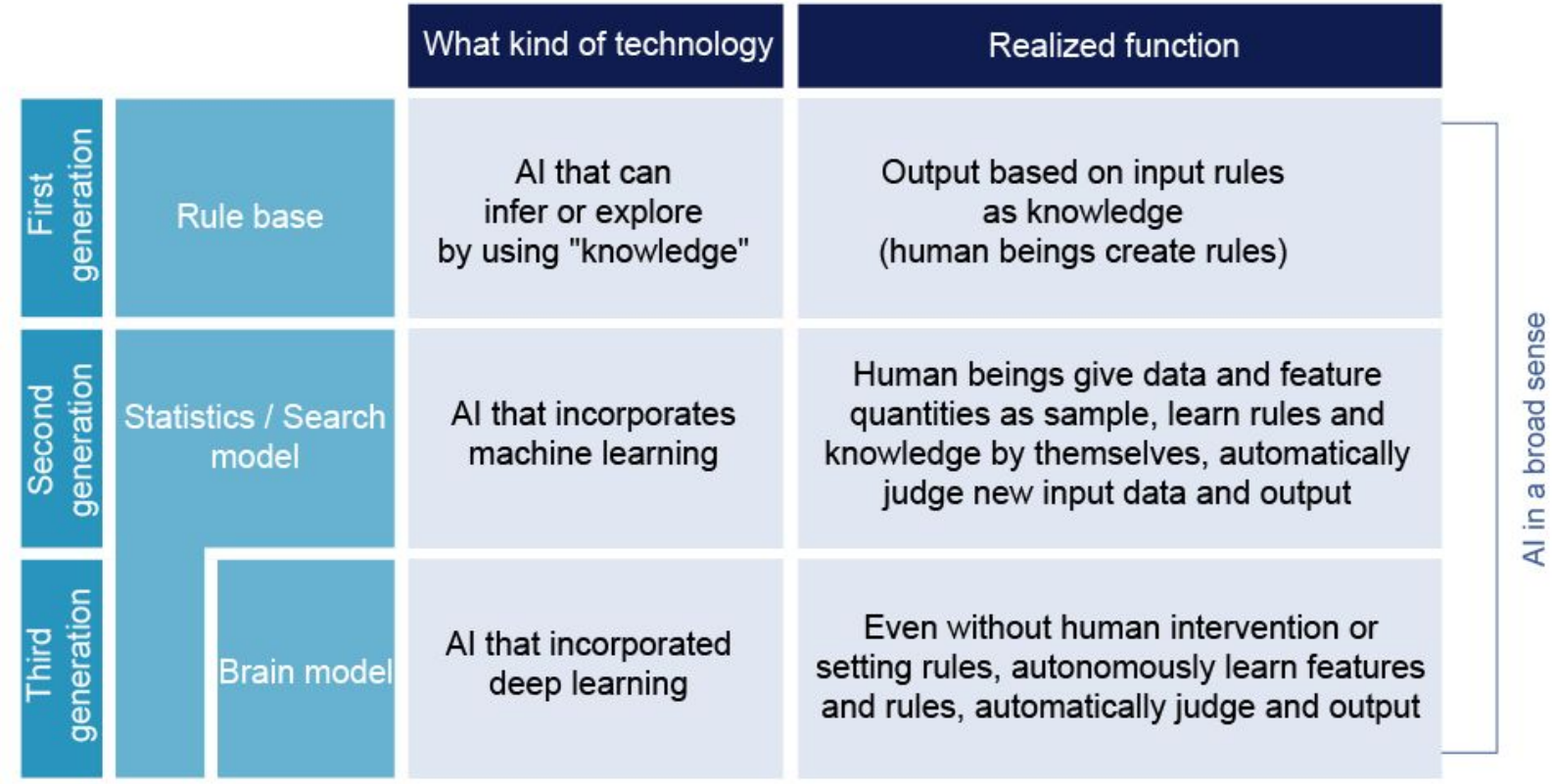
- Understanding
- Generation
- Translation

–Robot Control

AI PROBLEM AREAS/TASK



AI PROBLEM AREAS/TASK



IDENTIFY THE PRODUCT

Logical Agents

LOGICAL AGENT

Knowledge can be defined as the body of facts and principles accumulated by human-kind or the act, fact or state of knowing.

It includes having a familiarity with

–language, concepts,

–procedures, rules,

–ideas, abstractions ,

–places, customs,

–facts associations

–ability to use these notions effectively in modeling different aspects of the world

LOGICAL AGENT

- Represent the knowledge

in state-space

- Reason about the solution

in logical steps

KNOWLEDGE MODEL

• Facts

• Data is viewed as collection of disconnected facts.

–Example: It is raining.

• Information emerges when relationships among facts are established and understood; Provides answers to “who”, "what", "where“, and“ when”.

–Example : The temperature dropped 15 degrees and then it started raining.

• Knowledge emerges when relationships among patterns are identified and understood; Provides answers as "how".

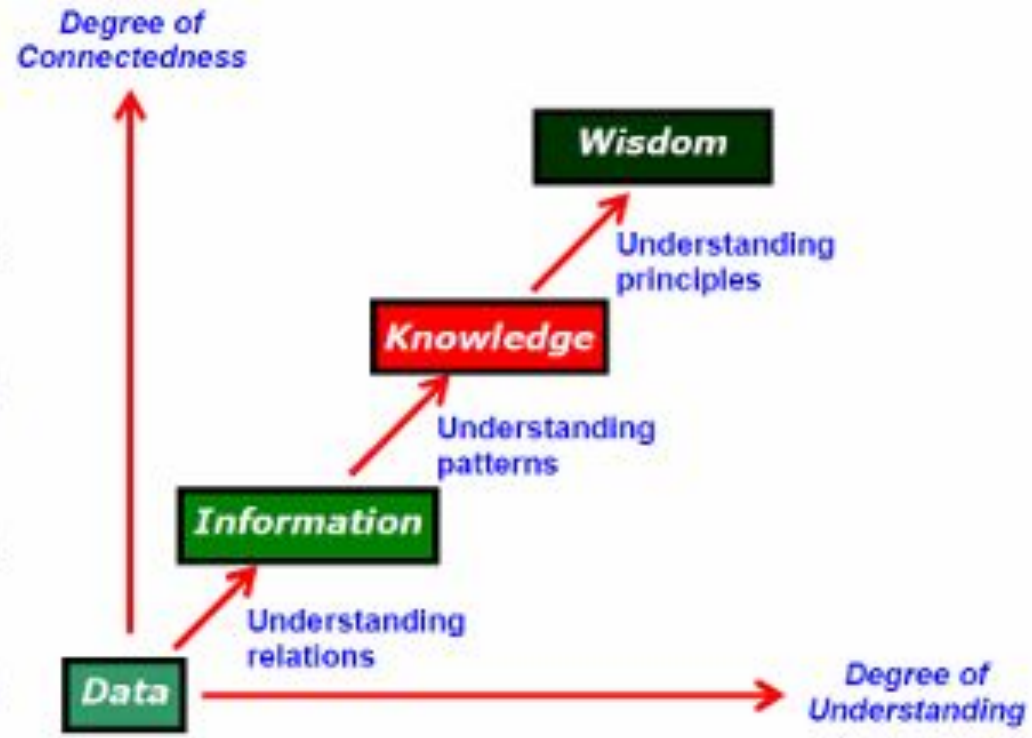
–Example : If the humidity is very high and the temperature drops substantially, then atmospheres is unlikely to hold the moisture, so it rains.

• Wisdom is the pinnacle of understanding, uncovers the principles of relationships that describe patterns. Provides answers as "why“

–Example: Encompasses understanding of all the interactions that happen between raining, evaporation, air currents, temperature gradients, changes, and raining

KNOWLEDGE MODEL

- Dont' confuse Knowledge with data.
- • Epistemology –study of nature of knowledge.
- • Metaknowldge –knowledge about knowledge



STATE SPACE REPRESENTATION

• In Search Strategies, the states are represented as

–Atomic representation

–Treated as black box

–Many factors are ignored, e.g., gas in car, money available for toll, etc.

• To include more factors in our state representation, we have

–Factored Representation

–Each state will be represented with

•Factors

•Each factor can be a Boolean or real-valued variable

•E.g., City Arad can now be represented as

•{“lat”: 46.1866, “lon”: 21.3123, “has_gas_till_next_station”: yes}

FACTORED REPRESENTATION

- Why do we need Factored representation

–To reason about steps

–To learn new knowledge about the environment

–To adapt to changes to the existing knowledge

–Accept new tasks in the form of explicit goals

–To overcome partial observability of environment

–E.g., Arad to Bucharest via Sibiu

–If Sibiu has road blocked, then the agent should take a different path

–i.e., to build Knowledge-based Agents

KNOWLEDGE BASED AGENTS

• Solving problems by

–Representing the knowledge in state-space–Reasoning about the solution in logical steps

• Central Component:

–Knowledge Base (KB)

• set of sentences, not the English sentences.,

–Represents some assertion about the world

• Sentence

–representation of knowledge in a language called Knowledge representation language

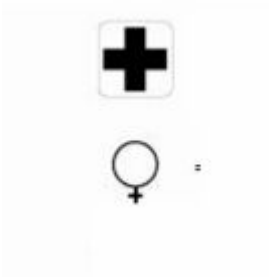
–Represents an axiom, when the sentence is taken as given without being derived from other sentences

• TELLoperation: Add new sentences to the knowledge base •

ASK operation: Query what is known

WHAT IS REPRESENTATION

- Symbols standing for things in the world



- Knowledge representation :

–symbolic encoding of preposition beloved (by some agent)

WHAT IS REASONING

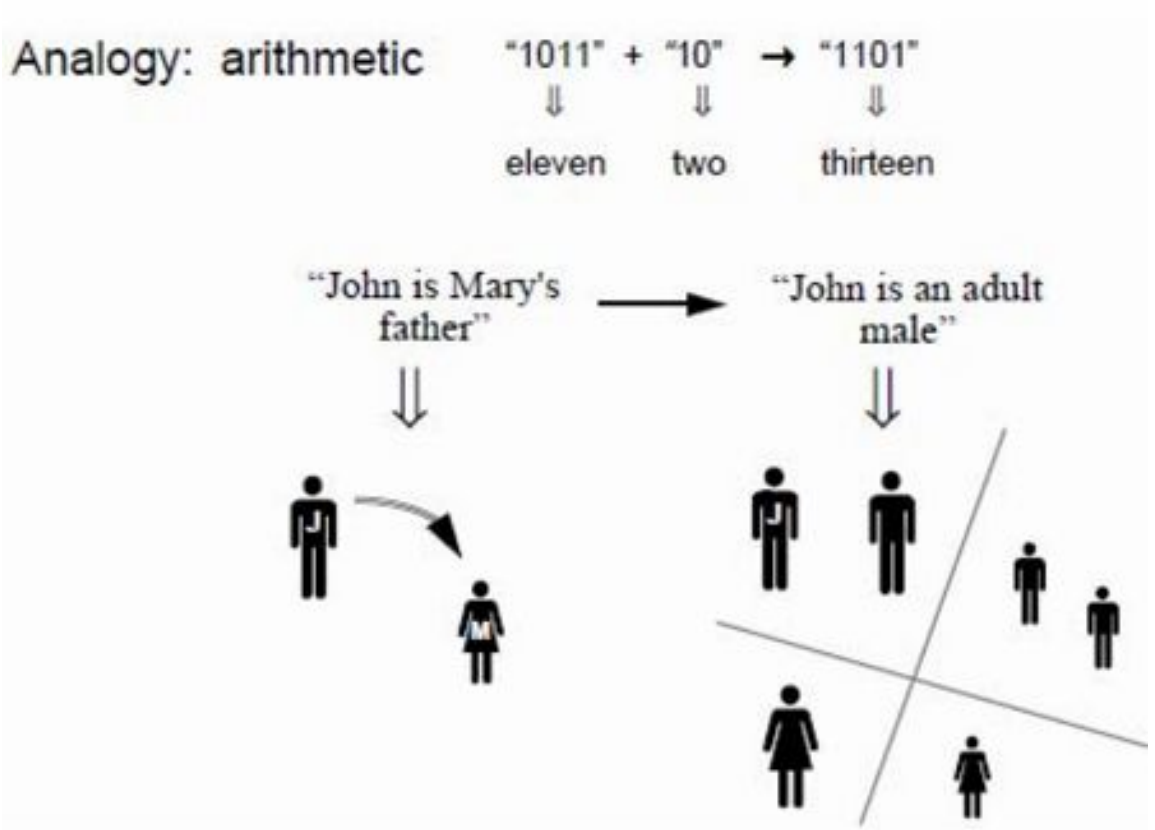
- Manipulation of symbols encoding propositions to produce representations of new propositions
- Benefits of Reasoning

–Given

- Patient X allergic to medication M
- Anyone allergic to medication M is also allergic to medication M'

–Reasoning helps us derive

- Patient X is allergic to medication M'



KNOWLEDGE REPRESENTATION AGENTS

- Each time a knowledge-based agent is called, it does three things

–TELL: the Knowledge Base about the percepts (inputs)

–ASK: the Knowledge Base what action it should perform.

•Extensive reasoning about the outcomes of possible action and current state

–TELL : the Knowledge Base about selected action (to update) and agent executes the action

• The agent must

–Represent states and actions

–Incorporate new percepts

–Update internal representations of the world

–Deduce hidden properties of the world

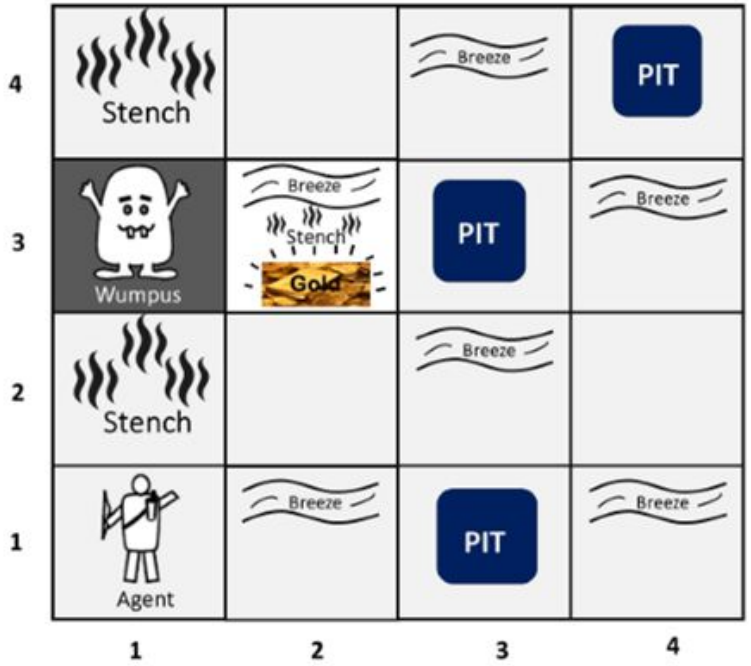
–Deduce appropriate action

KNOWLEDGE BASED AGENTS: WUMPUS WORLD

- The Wumpus world is a simple world example to illustrate the worth of a knowledge-based agent and to represent knowledge representation. It was inspired by a video game Hunt the Wumpus by Gregory Yob in 1973.
- The Wumpus world is a cave which has 4/4 rooms connected with passageways. So there are total 16 rooms which are connected with each other. We have a knowledge-based agent who will go forward in this world. The cave has a room with a beast which is called Wumpus, who eats anyone who enters the room. The Wumpus can be shot by the agent, but the agent has a single arrow. In the Wumpus world, there are some Pits rooms which are bottomless, and if agent falls in Pits, then he will be stuck there forever. The exciting thing with this cave is that in one room there is a possibility of finding a heap of gold. So the agent goal is to find the gold and climb out the cave without fallen into Pits or eaten by Wumpus. The agent will get a reward if he comes out with gold, and he will get a penalty if eaten by Wumpus or falls in the pit.

HERE WUMPUS IS STATIC AND CANNOT MOVE.

- Following is a sample diagram for representing the Wumpus world. It is showing some rooms with Pits, one room with Wumpus and one agent at (1, 1) square location of the world.



WUMPUS WORLD

- There are also some components which can help the agent to navigate the cave. These components are given as follows:
- The rooms adjacent to the Wumpus room are smelly, so that it would have some stench.
- The room adjacent to PITs has a breeze, so if the agent reaches near to PIT, then he will perceive the breeze.
- There will be glitter in the room if and only if the room has gold.
- The Wumpus can be killed by the agent if the agent is facing to it, and Wumpus will emit a horrible scream which can be heard anywhere in the cave.

PEAS DESCRIPTION OF WUMPUS WORLD

- Performance measure:

+1000 reward points if the agent comes out of the cave with the gold.

-1000 points penalty for being eaten by the Wumpus or falling into the pit.

-1 for each action, and -10 for using an arrow.

The game ends if either agent dies or came out of the cave.

PEAS

- Environment:

A 4*4 grid of rooms.

The agent initially in room square [1, 1], facing toward the right.

Location of Wumpus and gold are chosen randomly except the first square [1,1].

Each square of the cave can be a pit with probability 0.2 except the first square.

- Actuators:

Left turn, Right turn, Move forward

Grab, Release, Shoot.

PEAS

- Sensors:

The agent will perceive the **stench** if he is in the room adjacent to the Wumpus. (Not diagonally).

The agent will perceive **breeze** if he is in the room directly adjacent to the Pit.

The agent will perceive the **glitter** in the room where the gold is present.

The agent will perceive the **bump** if he walks into a wall.

When the Wumpus is shot, it emits a horrible **scream** which can be perceived anywhere in the cave.

These percepts can be represented as five element list, in which we will have different indicators for each sensor.

Example if agent perceives stench, breeze, but no glitter, no bump, and no scream then it can be represented as:
[Stench, Breeze, None, None, None].

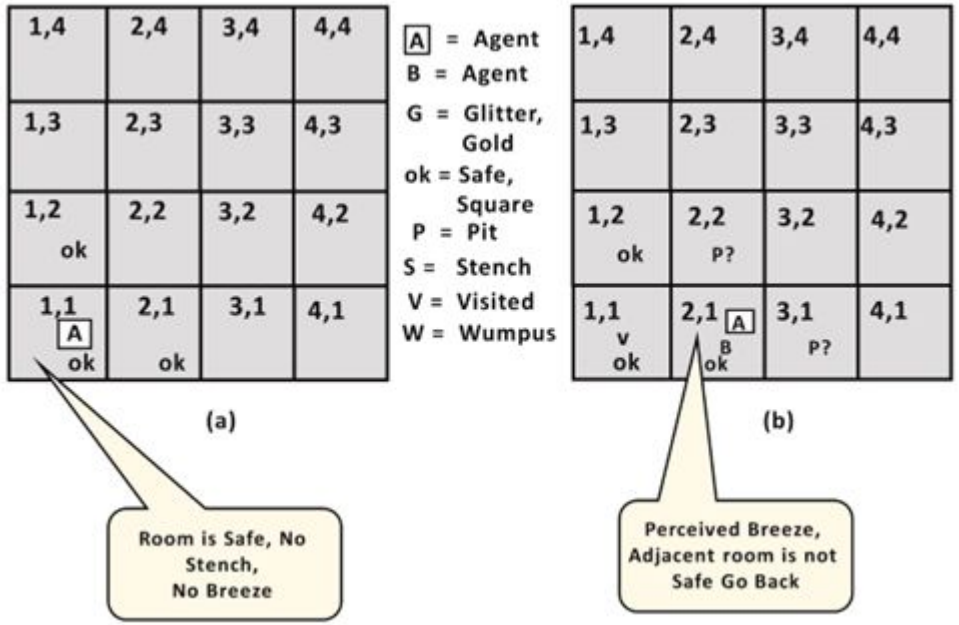
PROPERTIES OF WUMPUS WORLD

- Partially observable: The Wumpus world is partially observable because the agent can only perceive the close environment such as an adjacent room.
- Deterministic: It is deterministic, as the result and outcome of the world are already known.
- Sequential: The order is important, so it is sequential.
- Static: It is static as Wumpus and Pits are not moving.
- Discrete: The environment is discrete.
- One agent: The environment is a single agent as we have one agent only and Wumpus is not considered as an agent.

EXPLORING WUMPUS WORLD

- Agent's First step:

Initially, the agent is in the first room or on the square [1,1], and we already know that this room is safe for the agent, so to represent on the below diagram (a) that room is safe we will add symbol OK. Symbol A is used to represent agent, symbol B for the breeze, G for Glitter or gold, V for the visited room, P for pits, W for Wumpus.



At Room [1,1] agent does not feel any breeze or any Stench which means the adjacent squares are also OK.

EXPLORING WUMPUS WORLD

- Agent's second Step:

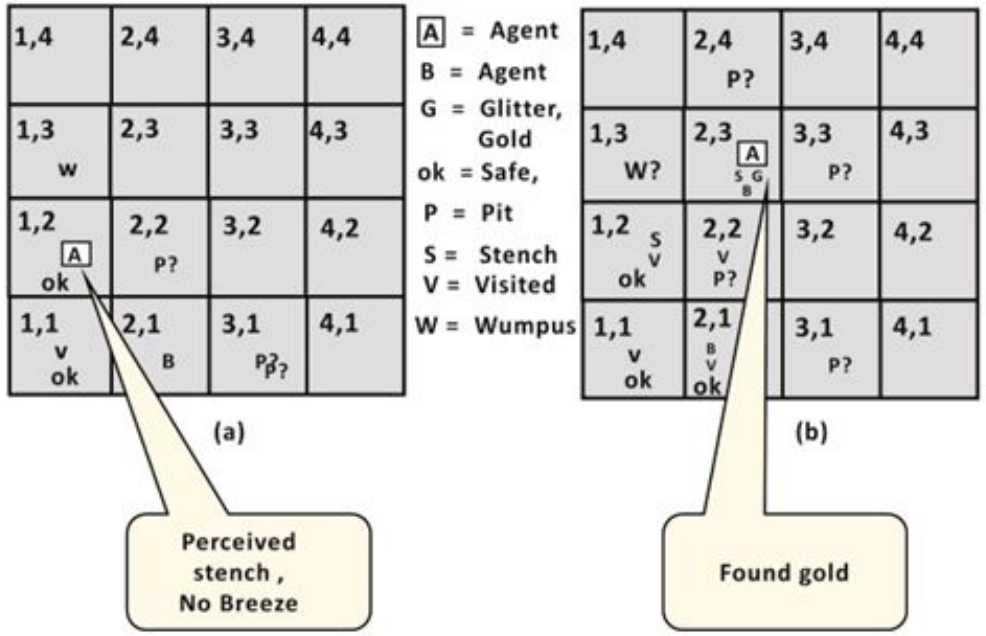
Now agent needs to move forward, so it will either move to [1, 2], or [2,1]. Let's suppose agent moves to the room [2, 1], at this room agent perceives some breeze which means Pit is around this room. The pit can be in [3, 1], or [2,2], so we will add symbol P? to say that, is this Pit room?

Now agent will stop and think and will not make any harmful move. The agent will go back to the [1, 1] room. The room [1,1], and [2,1] are visited by the agent, so we will use symbol V to represent the visited squares.

EXPLORING WUMPUS WORLD

- Agent's 3rd Step

At the third step, now agent will move to the room [1,2] which is OK. In the room [1,2] agent perceives a stench which means there must be a Wumpus nearby. But Wumpus cannot be in the room [1,1] as by rules of the game, and also not in [2,2] (Agent had not detected any stench when he was at [2,1]). Therefore agent infers That Wumpus is in the room [1,3], and in current state, there is no breeze which means in [2,2] there is no Pit and no Wumpus. So it is safe, and we will mark it OK, and the agent moves further in [2,2].



EXPLORING WUMPUS WORLD

- Agent's fourth step:

At room [2,2], here no stench and no breezes present so let's suppose agent decides to move to [2,3]. At room [2,3] agent perceives glitter, so it should grab the gold and climb out of the cave.

PROPOSITIONAL LOGIC IN ARTIFICIAL INTELLIGENCE

Propositional logic, also known as propositional calculus or sentential logic, forms the foundation of logical reasoning in artificial intelligence (AI). It is a branch of logic that deals with propositions, which can either be true or false. In AI, propositional logic is essential for knowledge representation, reasoning, and decision-making processes. This article delves into the fundamental concepts of propositional logic and its applications in AI.

PROPOSITIONAL LOGIC IN AI

- Propositional Logic is a kind of logic whereby the expression that takes into consideration is referred to as a proposition, which is a statement that can be either true or false but cannot be both at the same time. In AI propositions are those facts, conditions, or any other assertion regarding a particular situation or fact in the world. Propositional logic uses propositional symbols, connective symbols, and parentheses to build up propositional logic expressions otherwise referred to as propositions.
- Proposition operators like conjunction ($\wedge$), disjunction ($\vee$), negation $\neg$, implication $\rightarrow$, and biconditional $\leftrightarrow$ enable a proposition to be manipulated and combined in order to represent the underlying logical relations and rules.

EXAMPLE OF PROPOSITIONAL LOGIC

In propositional logic, well-formed formulas, also called propositions, are declarative statements that may be assigned a truth value of either true or false. They are often denoted by letters such as P, Q, and R. Here are some examples:

- P: In this statement, ‘The sky is blue’ five basic sentence components are used.
- Q: ‘There is only one thing wrong at the moment we are in the middle of a rain.”
- R: ‘Sometimes they were just saying things without realizing: “The ground is wet”’.

All these protasis can be connected by logical operations to create stamata with greater propositional depth. For instance:

- P⋀Q: ”It is clear that the word ‘nice’ for the sentence ‘Saturday is a nice day’ exists as well as the word ‘good’ for the sentence ‘The weather is good today. ’”
- P⋁Q: “It may probably be that the sky is blue or that it is raining. ”
- ¬P: I was not mindful that the old adage “The sky is not blue” deeply describes a geek.

BASIC CONCEPTS OF PROPOSITIONAL LOGIC

- 1. Propositions:
- A proposition is a declarative statement that is either true or false. For example:
- “The sky is blue.” (True)
- “It is raining.” (False)

BASIC CONCEPTS OF PROPOSITIONAL LOGIC

- 2. Logical Connectives:
- Logical connectives are used to form complex propositions from simpler ones. The primary connectives are:
- AND (∧): A conjunction that is true if both propositions are true.
  - Example: "It is sunny ∧ It is warm" is true if both propositions are true.
- OR (∨): A disjunction that is true if at least one proposition is true.
  - Example: "It is sunny ∨ It is raining" is true if either proposition is true.
- NOT (¬): A negation that inverts the truth value of a proposition.
  - Example: "¬It is raining" is true if "It is raining" is false.
- IMPLIES (→): A conditional that is true if the first proposition implies the second.
  - Example: "If it rains, then the ground is wet" (It rains → The ground is wet) is true unless it rains and the ground is not wet.
- IFF (↔): A biconditional that is true if both propositions are either true or false together.
  - Example: "It is raining ↔ The ground is wet" is true if both are true or both are false.

BASIC CONCEPTS OF PROPOSITIONAL LOGIC

- 3. Truth Tables:
- Truth tables are used to determine the truth value of complex propositions based on the truth values of their components. They exhaustively list all possible truth value combinations for the involved propositions.
- 4. Tautologies, Contradictions, and Contingencies:
- Tautology: A proposition that is always true, regardless of the truth values of its components.
  - Example: “P ∨ ¬P”
- Contradiction: A proposition that is always false.
  - Example: “P ∧ ¬P”
- Contingency: A proposition that can be either true or false depending on the truth values of its components.
  - Example: “P ∧ Q”

FACTS ABOUT PROPOSITIONAL LOGIC

- Bivalence: A proposition gives a true and false result, with no in-between because h/p' cannot be true and false simultaneously.
- Compositionality: The general signification of truth value of the proposition depends on the truth values of the parts that make up the proposition as well as the relations between the different parts.
- Non-ambiguity: Every purpose is unambiguous, well-defined: Each proposition is a well-defined purpose, which means that at any given moment there is only one possible interpretation of it.

SYNTAX OF PROPOSITIONAL LOGIC

- Propositional Logic and its syntax describes systems of propositions and methods for constructing well-formed propositions and statements. The main components include:
- Propositions: Denoted by capital letters (For example, P, Q).
- Logical Connectives: Signs that are employed to join give propositions (e.g., $\wedge$, $\vee$, $\neg$).
- Parentheses: Conventional operators are employed to identify the sequence of operations and the hierarchy of various operators existing in the syntax of computer programming languages.
- In propositional logic, a well-formed formula or WFF is an expression in symbols for the logic that satisfies the grammar rules of the logic.

LOGICAL EQUIVALENCE

- Two statements have the same logical form if the truth of every proposition contained in the first statement has the same value in all cases as the truth of every proposition contained in the second statement. For instance:
- It is also important to note that $S \rightarrow T$ is equivalent to $\neg S \vee T$.
- The deep relationship between $P \leftrightarrow Q$ and $(P \rightarrow Q) \wedge (Q \rightarrow P)$ can be easily deduced, and the relationship between $P \leftrightarrow Q$ and $(P \rightarrow Q) \wedge (Q \rightarrow P)$ cannot be overemphasized.
- Logical equivalence can be done using truth tables or logical equivalences where specific attributes are used to compare the two.

PROPERTIES OF OPERATORS

- The logical operators in propositional logic have several important properties:
- 1. Commutativity:
- $P \land Q \equiv Q \land P$
- $P \lor Q \equiv Q \lor P$
- 2. Associativity:
- $(P \land Q) \land R \equiv P \land (Q \land R)$
- $(P \lor Q) \lor R \equiv P \lor (Q \lor R)$
- 3. Distributivity:
- $P \land (Q \lor R) \equiv (P \land Q) \lor (P \land R)$
- $P \lor (Q \land R) \equiv (P \lor Q) \land (P \lor R)$

PROPERTIES OF OPERATORS

- 4. Identity:
- $P \wedge true \equiv P$
- $P \vee false \equiv P$
- 5. Domination:
- $P \vee true \equiv true$
- $P \wedge false \equiv false$
- 6. Double Negation:
- $\neg (\neg P) \equiv P$
- 7. Idempotence:
- $P \wedge P \equiv P$
- $P \vee P \equiv P$

First-Order Logic in Artificial intelligence

In the topic of Propositional logic, we have seen that how to represent statements using propositional logic. But unfortunately, in propositional logic, we can only represent the facts, which are either true or false. PL is not sufficient to represent the complex sentences or natural language statements. The propositional logic has very limited expressive power. Consider the following sentence, which we cannot represent using PL logic.

- "Some humans are intelligent", or
- "Sachin likes cricket."

To represent the above statements, PL logic is not sufficient, so we required some more powerful logic, such as first-order logic.

First Order Logic

- First-order logic is another way of knowledge representation in artificial intelligence. It is an extension to propositional logic.
- FOL is sufficiently expressive to represent the natural language statements in a concise way.
- First-order logic is also known as Predicate logic or First-order predicate logic. First-order logic is a powerful language that develops information about the objects in a more easy way and can also express the relationship between those objects.
- First-order logic (like natural language) does not only assume that the world contains facts like propositional logic but also assumes the following things in the world:
  - Objects: A, B, people, numbers, colors, wars, theories, squares, pits, wumpus, ...
  - Relations: It can be unary relation such as: red, round, is adjacent, or n-any relation such as: the sister of, brother of, has color, comes between
  - Function: Father of, best friend, third inning of, end of, ...
- As a natural language, first-order logic also has two main parts:
  - Syntax
  - Semantics

Syntax of First Order Logic and Basic Elements

The syntax of FOL determines which collection of symbols is a logical expression in first-order logic. The basic syntactic elements of first-order logic are symbols. We write statements in short-hand notation in FOL.

Constant	1, 2, A, John, Mumbai, cat,....
Variables	x, y, z, a, b,....
Predicates	Brother, Father, >,....
Function	sqrt, LeftLegOf, ...
Connectives	∧, ∨, ¬, ⇒, ⇔
Equality	==
Quantifier	∀, ∃

Atomic Sentences and Complex Sentences

- Atomic sentences are the most basic sentences of first-order logic. These sentences are formed from a predicate symbol followed by a parenthesis with a sequence of terms.
- We can represent atomic sentences as Predicate (term1, term2, ..., term n).

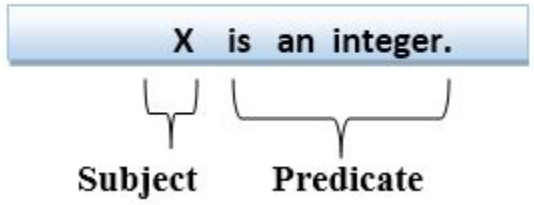
Example: Ravi and Ajay are brothers: => Brothers(Ravi, Ajay).
Chinky is a cat: => cat (Chinky).

- Complex sentences are made by combining atomic sentences using connectives.

First-order logic statements can be divided into two parts:

- Subject: Subject is the main part of the statement.
- Predicate: A predicate can be defined as a relation, which binds two atoms together in a statement.

Consider the statement: "x is an integer," it consists of two parts, the first part x is the subject of the statement and second part "is an integer," is known a



Quantifiers in First-order logic:

- A quantifier is a language element which generates quantification, and quantification specifies the quantity of specimen in the universe of discourse.
- These are the symbols that permit to determine or identify the range and scope of the variable in the logical expression. There are two types of quantifier:
  1. **Universal Quantifier, (for all, everyone, everything)**
  2. **Existential quantifier, (for some, at least one).**

Universal Quantifier:

Universal quantifier is a symbol of logical representation, which specifies that the statement within its range is true for everything or every instance of a particular thing.

The Universal quantifier is represented by a symbol ∀, which resembles an inverted A.

If x is a variable, then ∀x is read as:

- **For all x**
- **For each x**
- **For every x.**

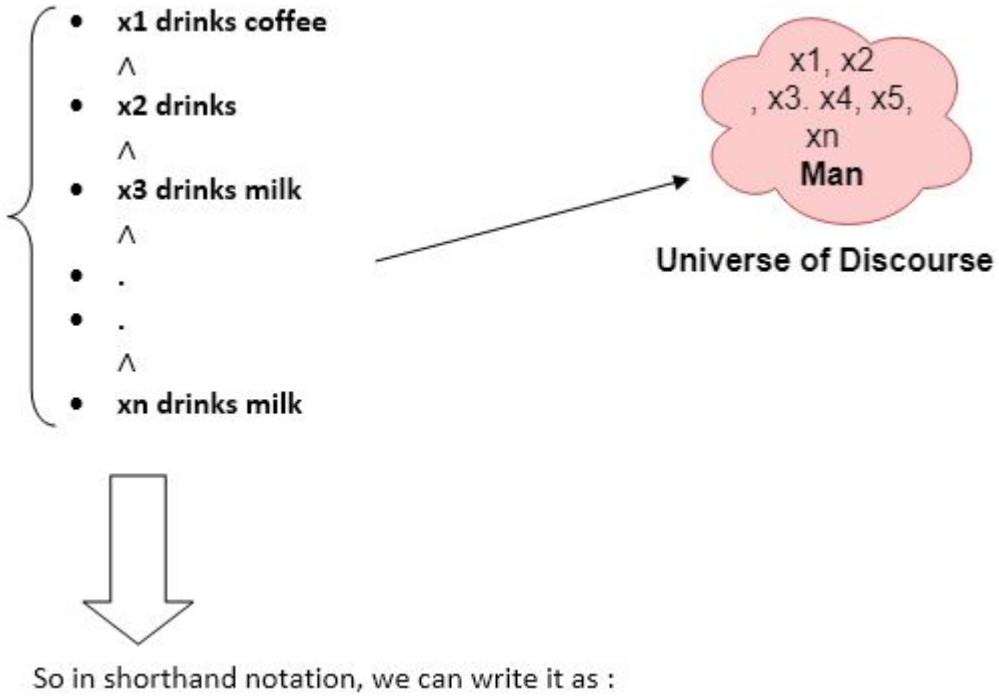
Example:

All man drink coffee.

Let a variable x which refers to a cat so all x can be represented in UOD as below:

∀x man(x) → drink (x, coffee).

It will be read as: There are all x where x is a man who drink coffee.



Existential Quantifier:

Existential quantifiers are the type of quantifiers, which express that the statement within its scope is true for at least one instance of something.

It is denoted by the logical operator ∃, which resembles as inverted E. When it is used with a predicate variable then it is called as an existential quantifier.

If x is a variable, then existential quantifier will be ∃x or ∃(x). And it will be read as:

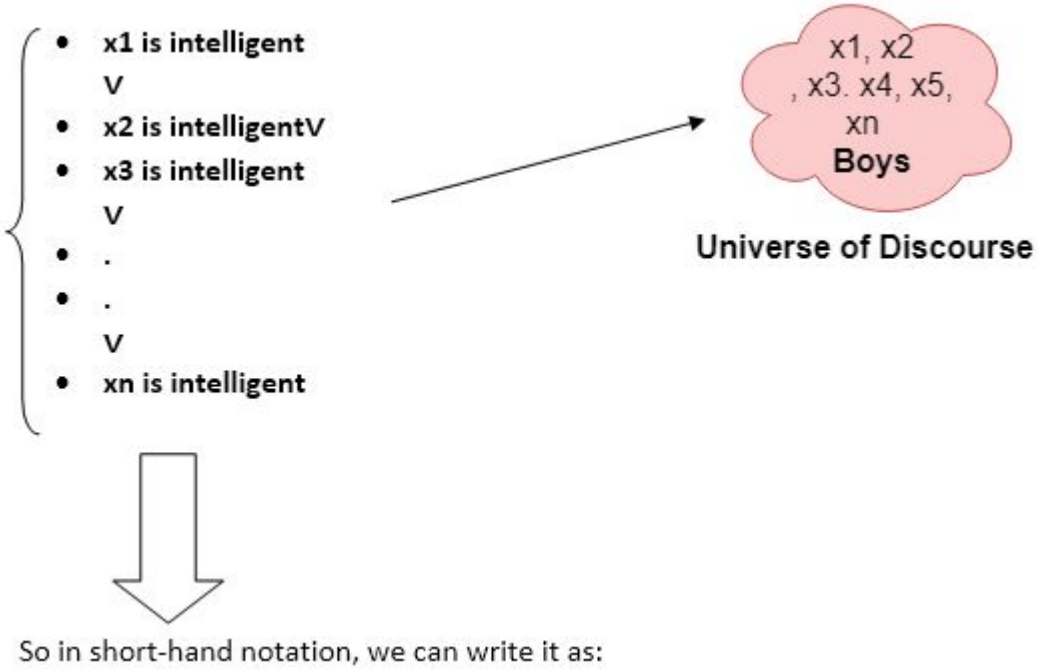
- **There exists a 'x.'**
- **For some 'x.'**
- **For at least one 'x.'**

Example:

Some boys are intelligent.

∃x: boys(x) ∧ intelligent(x)

It will be read as: There are some x where x is a boy who is intelligent.



Points to remember:

- The main connective for universal quantifier ∀ is implication →.
- The main connective for existential quantifier ∃ is and ∧.

Properties of Quantifiers:

- In universal quantifier, ∀x∀y is similar to ∀y∀x.
- In Existential quantifier, ∃x∃y is similar to ∃y∃x.
- ∃x∀y is not similar to ∀y∃x.

Some Examples of FOL using quantifier:

1. All birds fly.
In this question the predicate is "**fly(bird)**."
And since there are all birds who fly so it will be represented as follows.

∀x bird(x) →fly(x).

2. Every man respects his parent.
In this question, the predicate is "**respect(x, y)," where x=man, and y= parent**.
Since there is every man so will use ∀, and it will be represented as follows:

∀x man(x) → respects (x, parent).

3. Some boys play cricket.
In this question, the predicate is "**play(x, y)**," where x= boys, and y= game. Since there are some boys so we will use **∃, and it will be represented as**:

∃x boys(x) → play(x, cricket).

4. Not all students like both Mathematics and Science.
In this question, the predicate is "**like(x, y)," where x= student, and y= subject**.
Since there are not all students, so we will use **∀ with negation, so** following representation for this:

¬∀ (x) [student(x) → like(x, Mathematics) ∧ like(x, Science)].

5. Only one student failed in Mathematics.
In this

question, the predicate is "**failed(x, y)," where x= student, and y= subject**.
Since there is only one student who failed in Mathematics, so we will use following representation for this:

∃ (x) [student(x) → failed (x, Mathematics) ∧∀ (y) [¬(x==y) ∧ student(y) → ¬failed (x, Mathematics)].

Free and Bound Variables:

The quantifiers interact with variables which appear in a suitable way. There are two types of variables in First-order logic which are given below:

Free Variable: A variable is said to be a free variable in a formula if it occurs outside the scope of the quantifier.

Example: ∀x ∃(y)[P (x, y, z)], where z is a free variable.

Bound Variable: A variable is said to be a bound variable in a formula if it occurs within the scope of the quantifier.

Example: ∀x [A (x) B(y)], here x and y are the bound variables.

Forward Chaining and Backward Chaining

In artificial intelligence, forward and backward chaining is one of the important topics, but before understanding forward and backward chaining let's first understand that from where these two terms came.

The inference engine is the component of the intelligent system in artificial intelligence, which applies logical rules to the knowledge base to infer new information from known facts. The first inference engine was part of the expert system. Inference engine commonly proceeds in two modes, which are:

1. Forward chaining
2. Backward chaining

Horn Clause and Definite clause:

Horn clause and definite clause are the forms of sentences, which enables knowledge base to use a more restricted and efficient inference algorithm. Logical inference algorithms use forward and backward chaining approaches, which require KB in the form of the first-order definite clause.

Definite clause: A clause which is a disjunction of literals with exactly one positive literal is known as a definite clause or strict horn clause.
Horn clause: A clause which is a disjunction of literals with at most one positive literal is known as horn clause. Hence all the definite clauses are horn clauses.

Example: (¬ p V ¬ q V k). It has only one positive literal k.

It is equivalent to p ∧ q → k.

Forward Chaining

Forward chaining is also known as a forward deduction or forward reasoning method when using an inference engine. Forward chaining is a form of reasoning which start with atomic sentences in the knowledge base and applies inference rules (Modus Ponens) in the forward direction to extract more data until a goal is reached.
The Forward-chaining algorithm starts from known facts, triggers all rules whose premises are satisfied, and add their conclusion to the known facts. This process repeats until the problem is solved.

Properties of Forward-Chaining:

- It is a down-up approach, as it moves from bottom to top.
- It is a process of making a conclusion based on known facts or data, by starting from the initial state and reaches the goal state.
- Forward-chaining approach is also called as data-driven as we reach to the goal using available data.
- Forward -chaining approach is commonly used in the expert system, such as CLIPS, business, and production rule systems.

Example

"As per the law, it is a crime for an American to sell weapons to hostile nations. Country A, an enemy of America, has some missiles, and all the missiles were sold to it by Robert, who is an American citizen."
Prove that "Robert is criminal."

To solve the above problem, first, we will convert all the above facts into first-order definite clauses, and then we will use a forward-chaining algorithm to reach the goal.

Facts Conversion into FOL:

It is a crime for an American to sell weapons to hostile nations. (Let's say p, q, and r are variables)

American (p) ∧ weapon(q) ∧ sells (p, q, r) ∧ hostile(r) → Criminal(p) ...(1)

Country A has some missiles. ?p Owns(A, p) ∧ Missile(p). It can be written in two definite clauses by using Existential Instantiation, introducing new Constant T1.

Owns(A, T1) .....(2)

Missile(T1) .....(3)

All of the missiles were sold to country A by Robert.

?p Missiles(p) ∧ Owns (A, p) → Sells (Robert, p, A) .....(4)

Missiles are weapons.

Missile(p) → Weapons (p) .....(5)

Facts Conversion into FOL:

It is a crime for an American to sell weapons to hostile nations. (Let's say p, q, and r are variables)

American (p) ∧ weapon(q) ∧ sells (p, q, r) ∧ hostile(r) → Criminal(p) ...(1)

Country A has some missiles. ?p Owns(A, p) ∧ Missile(p). It can be written in two definite clauses by using Existential Instantiation, introducing new Constant T1.

Owns(A, T1) .....(2)

Missile(T1) .....(3)

All of the missiles were sold to country A by Robert.

?p Missiles(p) ∧ Owns (A, p) → Sells (Robert, p, A) .....(4)

Missiles are weapons.

Missile(p) → Weapons (p) .....(5)

Enemy of America is known as hostile.

Enemy(p, America) →Hostile(p) .....(6)

Country A is an enemy of America.

Enemy (A, America) .....(7)

Robert is American

American(Robert). .....(8)

Forward chaining proof:

Step-1:

In the first step we will start with the known facts and will choose the sentences which do not have implications, such as: American(Robert), Enemy(A, America), Owns(A, T1), and Missile(T1). All these facts will be represented as below.

Step-2:

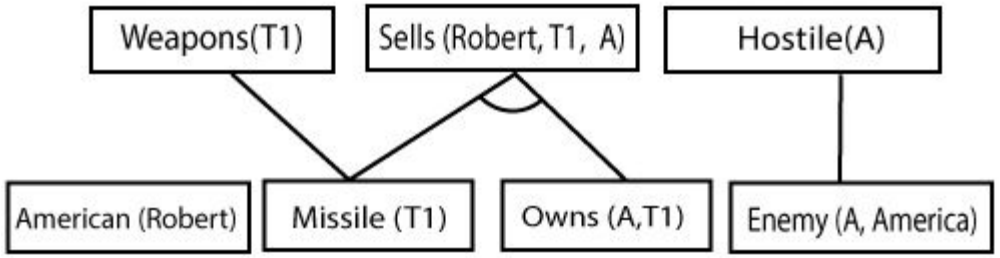
At the second step, we will see those facts which infer from available facts and with satisfied premises.

Rule-(1) does not satisfy premises, so it will not be added in the first iteration.

Rule-(2) and (3) are already added.

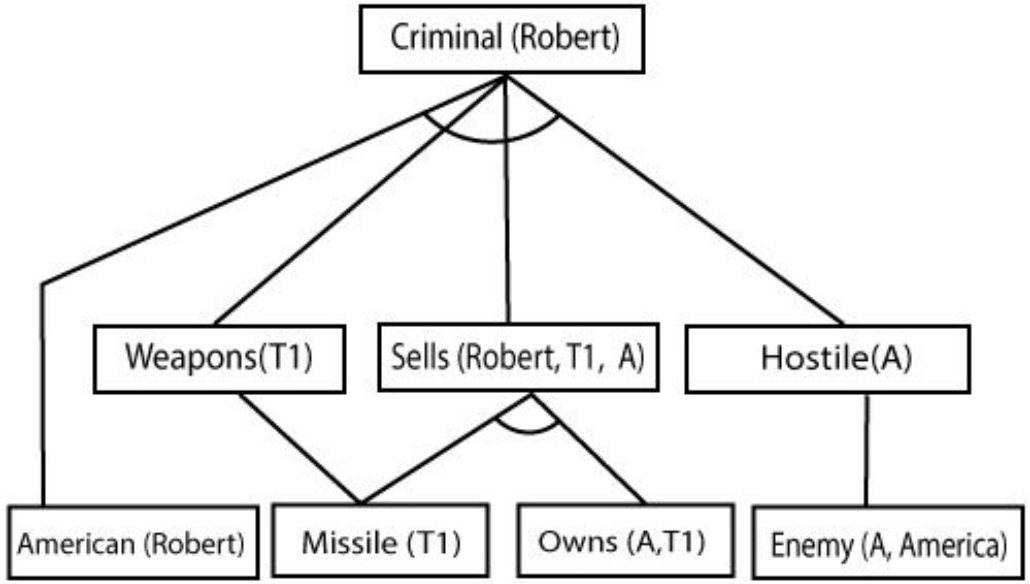
Rule-(4) satisfy with the substitution {p/T1}, so Sells (Robert, T1, A) is added, which infers from the conjunction of Rule (2) and (3).

Rule-(6) is satisfied with the substitution(p/A), so Hostile(A) is added and which infers from Rule-(7).



Step 3:

At step-3, as we can check Rule-(1) is satisfied with the substitution {p/Robert, q/T1, r/A}, so we can add Criminal(Robert) which infers all the available facts. And hence we reached our goal statement.



Hence it is proved that Robert is Criminal using forward chaining approach.

Backward Chaining:

Backward-chaining is also known as a backward deduction or backward reasoning method when using an inference engine. A backward chaining algorithm is a form of reasoning, which starts with the goal and works backward, chaining through rules to find known facts that support the goal.

Properties of backward chaining:

- It is known as a top-down approach.
- Backward-chaining is based on modus ponens inference rule.
- In backward chaining, the goal is broken into sub-goal or sub-goals to prove the facts true.
- It is called a goal-driven approach, as a list of goals decides which rules are selected and used.
- Backward -chaining algorithm is used in game theory, automated theorem proving tools, inference engines, proof assistants, and various AI applications.
- The backward-chaining method mostly used a depth-first search strategy for proof.

Example

In backward-chaining, we will use the same above example, and will rewrite all the rules.

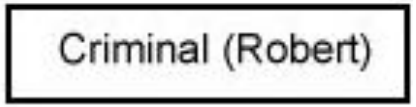
- American (p) ∧ weapon(q) ∧ sells (p, q, r) ∧ hostile(r) → Criminal(p) ...(1)
  Owns(A, T1) .....(2)
- Missile(T1)
- ?p Missiles(p) ∧ Owns (A, p) → Sells (Robert, p, A) .....(4)
- Missile(p) → Weapons (p) .....(5)
- Enemy(p, America) →Hostile(p) .....(6)
- Enemy (A, America) .....(7)
- American(Robert). .....(8)

Backward Chaining Proof

In Backward chaining, we will start with our goal predicate, which is Criminal(Robert), and then infer further rules.

Step-1:

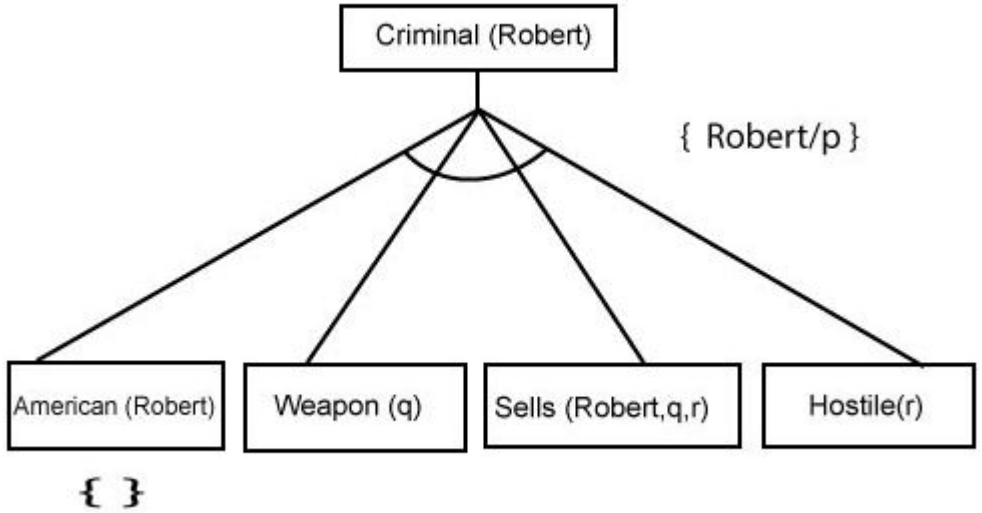
At the first step, we will take the goal fact. And from the goal fact, we will infer other facts, and at last, we will prove those facts true. So our goal fact is "Robert is Criminal," so following is the predicate of it.



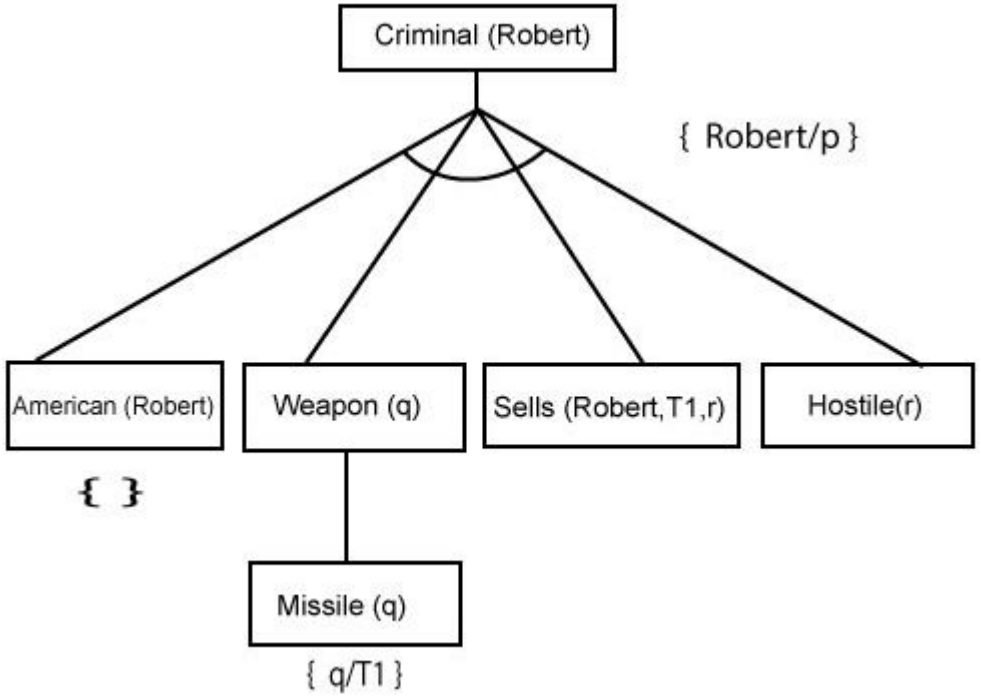
Step-2:

At the second step, we will infer other facts form goal fact which satisfies the rules. So as we can see in Rule-1, the goal predicate Criminal (Robert) is present with substitution {Robert/P}. So we will add all the conjunctive facts below the first level and will replace p with Robert.

Here we can see American (Robert) is a fact, so it is proved here.

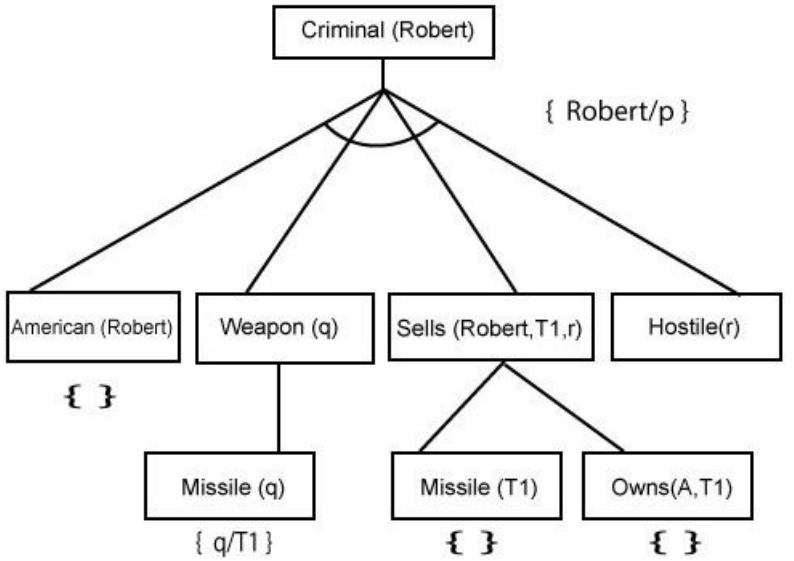


Step-3:t At step-3, we will extract further fact Missile(q) which infer from Weapon(q), as it satisfies Rule-(5). Weapon (q) is also true with the substitution of a constant T1 at q.



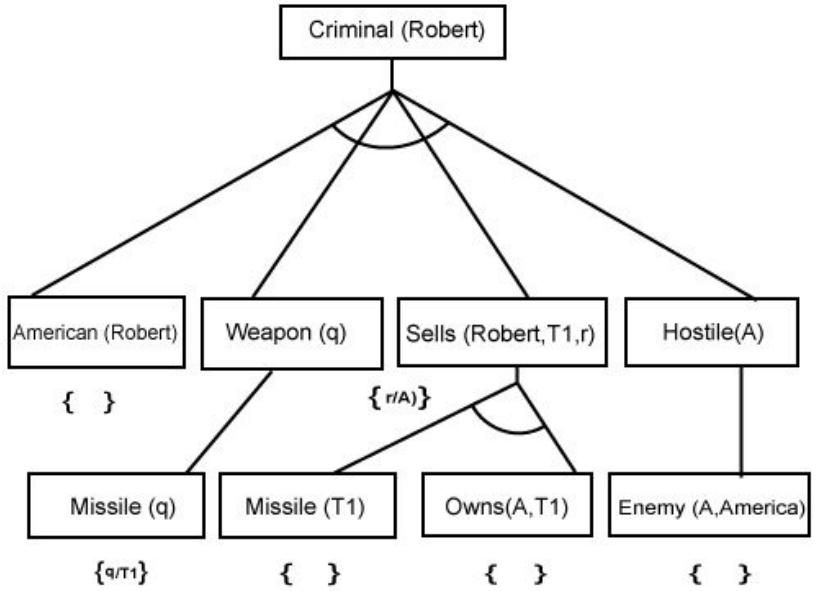
Step-4:

At step-4, we can infer facts Missile(T1) and Owns(A, T1) form Sells(Robert, T1, r) which satisfies the Rule- 4, with the substitution of A in place of r. So these two statements are proved here.



Step-5:

At step-5, we can infer the fact Enemy(A, America) from Hostile(A) which satisfies Rule- 6. And hence all the statements are proved true using backward chaining.



Difference

- Forward chaining as the name suggests, start from the known facts and move forward by applying inference rules to extract more data, and it continues until it reaches to the goal, whereas backward chaining starts from the goal, move backward by using inference rules to determine the facts that satisfy the goal.
- Forward chaining is called a data-driven inference technique, whereas backward chaining is called a goal-driven inference technique.
- Forward chaining is known as the down-up approach, whereas backward chaining is known as a top-down approach.
- Forward chaining uses breadth-first search strategy, whereas backward chaining uses depth-first search strategy.
- Forward and backward chaining both applies Modus ponens inference rule.
- Forward chaining can be used for tasks such as planning, design process monitoring, diagnosis, and classification, whereas backward chaining can be used for classification and diagnosis tasks.
- Forward chaining can be like an exhaustive search, whereas backward chaining tries to avoid the unnecessary path of reasoning.
- In forward-chaining there can be various ASK questions from the knowledge base, whereas in backward chaining there can be fewer ASK questions.
- Forward chaining is slow as it checks for all the rules, whereas backward chaining is fast as it checks few required rules only.

Basics of Prolog

Prolog or **PRO**gramming in **LOG**ics is a logical and declarative programming language. It is one major example of the fourth generation language that supports the declarative programming paradigm. This is particularly suitable for programs that involve **symbolic** or **non-numeric computation**.

This is the main reason to use Prolog as the programming language in **Artificial Intelligence**, where **symbol manipulation** and **inference manipulation** are the fundamental tasks.

In Prolog, we need not mention the way how one problem can be solved, we just need to mention what the problem is, so that Prolog automatically solves it. However, in Prolog we are supposed to give clues as the **solution method**.

Prolog language basically has three different elements –

Facts – The fact is predicate that is true, for example, if we say, “Tom is the son of Jack”, then this is a fact.

Rules – Rules are extinctions of facts that contain conditional clauses. To satisfy a rule these conditions should be met. For example, if we define a rule as –

grandfather(X, Y) :- father(X, Z), parent(Z, Y)

This implies that for X to be the grandfather of Y, Z should be a parent of Y and X should be father of Z.

Questions – And to run a prolog program, we need some questions, and those questions can be answered by the given facts and rules.

Prolog Applications

- Intelligent Database Retrieval
- Natural Language Understanding
- Specification Language
- Machine Learning
- Robot Planning
- Automation System
- Problem Solving

Prolog Examples

```
% Facts
man(john).
woman(mary).
capital_of(france, paris).
% Rule
not(X,Y) :- man(X), woman(Y).
% Query 1
not(john, mary)?
% Query 2
capital_of(france, X)?
```

Here is a simple Prolog program that defines a set of rules and facts about a problem domain, and then uses those rules and facts to answer a few queries:

Example explanation

In this example, the program defines three facts: man(john), woman(mary), and capital_of(france, paris). These facts state that John is a man, Mary is a woman and Paris is the capital of France.

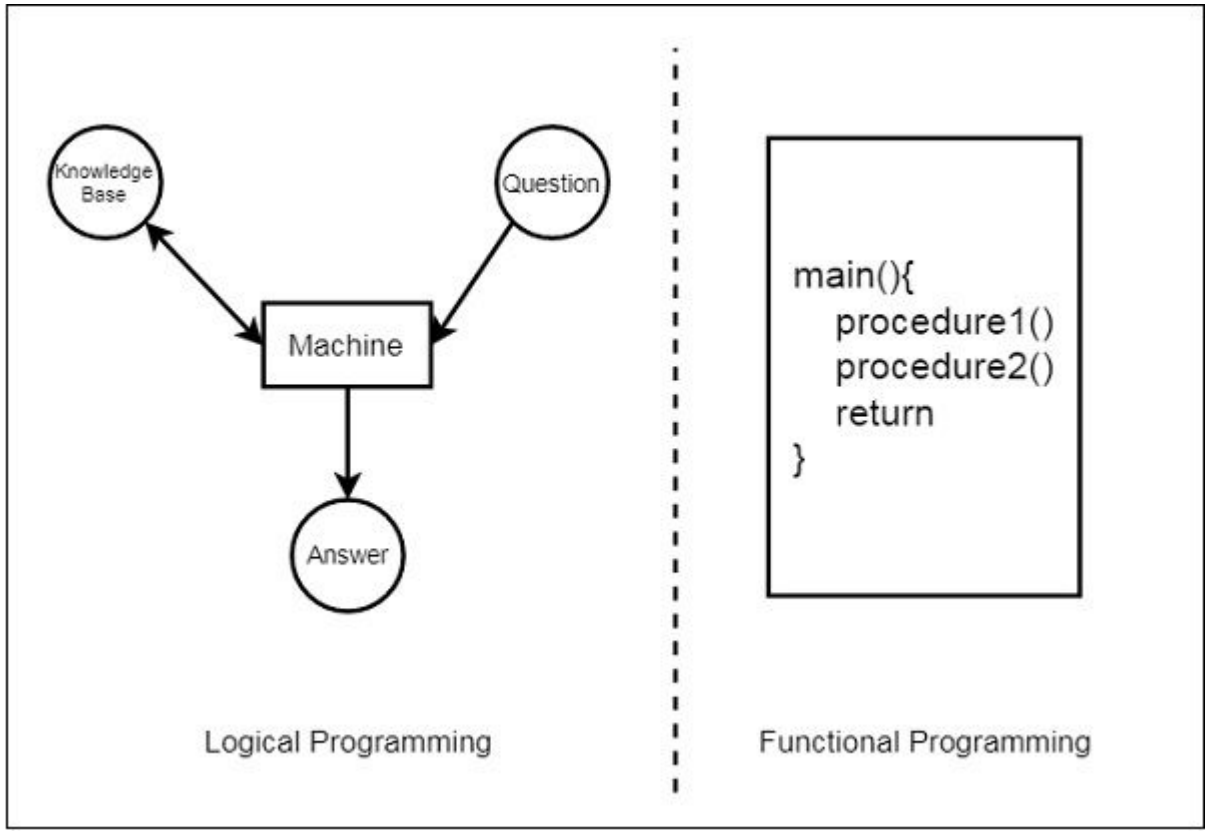
The program also defines a rule using the not/2 predicate. This rule states that if X is a man and Y is a woman, then X is not Y.

Finally, the program includes two queries. The first query, not(john, mary)?, asks the interpreter to determine whether John is not Mary, based on the not/2 rule and the man/1 and woman/1 facts. The interpreter will use these rules and facts to deduce that John is not Mary, and it will return true as the solution to the query.

The second query, capital_of(france, X)?, asks the interpreter to determine the capital of France. The interpreter will use the capital_of/2 fact to determine that the capital of France is Paris, and it will return the value paris for the variable X as the solution.

Overall, this Prolog program demonstrates how to define rules and facts about a problem domain, and how to use those rules and facts to automatically infer solutions to problems.

Logic and Functional Programming



Logic and Functional Programming

From this illustration, we can see that in Functional Programming, we have to define the procedures, and the rule how the procedures work. These procedures work step by step to solve one specific problem based on the algorithm. On the other hand, for the Logic Programming, we will provide knowledge base. Using this knowledge base, the machine can find answers to the given questions, which is totally different from functional programming.

In functional programming, we have to mention how one problem can be solved, but in logic programming we have to specify for which problem we actually want the solution. Then the logic programming automatically finds a suitable solution that will help us solve that specific problem.

Logic and Functional Programming

Functional Programming	Logic Programming
Functional Programming follows the Von-Neumann Architecture, or uses the sequential steps.	Logic Programming uses abstract model, or deals with objects and their relationships.
The syntax is actually the sequence of statements like (a, s, l).	The syntax is basically the logic formulae (Horn Clauses).
The computation takes part by executing the statements sequentially.	It computes by deducting the clauses.
Logic and controls are mixed together.	Logics and controls can be separated.

Thank You