ARITIFICAL INTELLIGENCE
MVJ22CS552

MODULE 2

Prepared by:
Mrs. POSHITHA.M
AP, Dept. of CSE,
MVJCE, Bangalore

Course objective

This course will give ability to:

- Understand fundamental concepts in Artificial Intelligence.
- Analyse problem-solving techniques and knowledge representation.
- Design intelligent components or programs to meet desired needs.
- Implement and evaluate computer-based intelligent systems.

Course Outcomes

At the end of the course, the student will be able to:

CO1: Understand fundamental concepts in Artificial Intelligence.
CO2: Analyze the problem-solving techniques and knowledge representation.
CO3: Design intelligent components or programs to meet desired needs.
CO4: Implement and evaluate computer-based intelligent systems.

Textbook and Reference books

Textbooks:

1.Stuart Russel, Peter Norvig, “Artificial Intelligence – A Modern Approach”, Pearson Education ,3rd Edition, 2009.

2.E.Rich and K.Knight, “Artificial Intelligence”, Tata McGraw Hill ,3rd Edition, ,2008.

3. Patterson, “Artificial Intelligence and Expert Systems”, PHI, 2nd Edition, 2009.

Reference Book:

1. Ivan Bratka, “PROLOG Programming for Artificial Intelligence”, Pearson Education, 3rd Edition, 2000.

MODULE 2

- **Knowledge Representation & Reasons:**

1. Knowledge Based Agents,
2. The Wumpus world.

- **Propositional Logic:**

1. Reasoning patterns in propositional logic, Resolution,
2. Forward & Backward Chaining, Inference in First order logic,
3. Propositional v/s first order inference,
4. Unification & lifting.

Knowledge Representation & Reasons

Humans are best at understanding, reasoning, and interpreting knowledge.

Human knows things, which is knowledge and as per their knowledge they perform various actions in the real world.

But how machines do all these things ?

This comes under knowledge representation and reasoning.

Hence we can describe Knowledge representation as following:

Knowledge representation and reasoning (KR, KRR) is the part of Artificial intelligence which concerned with “AI agents thinking and how thinking contributes to intelligent behavior of agents.”

- It is responsible for representing information about the real world so that a computer can understand and can utilize this knowledge to solve the complex real world problems such as diagnosis a medical condition or communicating with humans in natural language.
- Knowledge representation is not just storing data into some database, but it also enables an intelligent machine to learn from that knowledge and experiences so that it can behave intelligently like a human.

What to Represent

Following are the kind of knowledge which needs to be represented in AI systems:

- **Object:** All the facts about objects in our world domain. E.g., Guitars contains strings, trumpets are brass instruments.
- **Events:** Events are the actions which occur in our world.
- **Performance:** It describe behavior which involves knowledge about how to do things.
- **Meta-knowledge:** It is knowledge about what we know.
- **Facts:** Facts are the truths about the real world and what we represent.
- **Knowledge-Base:** The central component of the knowledge-based agents is the knowledge base. It is represented as KB. The Knowledgebase is a group of the Sentences (Here, sentences are used as a technical term and not identical with the English language).
- **Knowledge:** Knowledge is awareness or familiarity gained by experiences of facts, data, and situations.

Types of knowledge

Following are the various types of knowledge:

knowledge about the structure of a problem or system and is often used to help AI systems decompose complex problems into simpler sub-problems that can be solved more easily. It is the basic knowledge of problem-solving.

heuristic knowledge are a hypothesis, common sense, rule of thumb, and intuition.

Knowledge about how to solve a problem

1. Declarative Knowledge:

- Declarative knowledge is to know about something.
- It includes concepts, facts, and objects.
- It is also called descriptive knowledge and expressed in declarativesentences.
- It is simpler than procedural language.

2. Procedural Knowledge

- It is also known as imperative knowledge.
- Procedural knowledge is a type of knowledge which is responsible for knowing how to do something.
- It can be directly applied to any task.
- It includes rules, strategies, procedures, agendas, etc.
- Procedural knowledge depends on the task on which it can be applied.

3. Meta-knowledge:

- Knowledge about the other types of knowledge is called Meta-knowledge.

4. Heuristic knowledge:

- Heuristic knowledge is representing knowledge of some experts in a filed or subject.
- Heuristic knowledge is rules of thumb based on previous experiences, awareness of approaches, and which are good to work but not guaranteed.

5. Structural knowledge:

- Structural knowledge is basic knowledge to problem-solving.
- It describes relationships between various concepts such as kind of, part of, and grouping of something.
- It describes the relationship that exists between concepts or objects.

AI knowledge cycle:

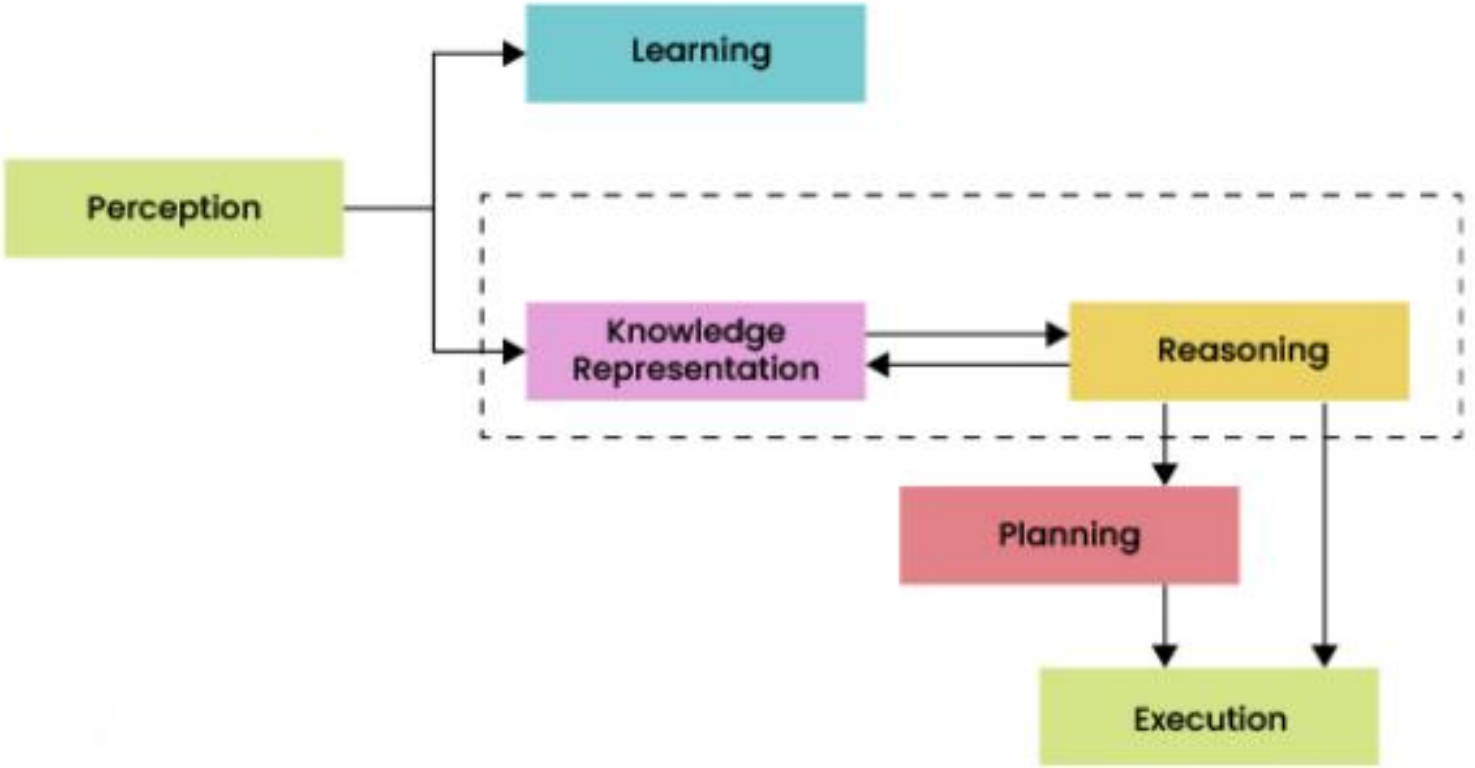
The relation between knowledge and intelligence:

- Knowledge of real-worlds plays a vital role in intelligence and same for creating artificial intelligence. Knowledge plays an important role in demonstrating intelligent behavior in AI agents.

An Artificial intelligence system has the following components for displaying intelligent behavior:

1. Perception
2. Learning
3. Knowledge Representation and Reasoning
4. Planning
5. Execution

AI Knowledge Cycle



- The above diagram is showing how an AI system can interact with the real world and what components help it to show intelligence.
- AI system has Perception component by which it retrieves information from its environment. It can be visual, audio or another form of sensory input. The learning component is responsible for learning from data captured by Perception comportment.
- In the complete cycle, **the main components are knowledge representation and Reasoning.**
- These two components are involved in showing the intelligence in machine-like humans. These two components are independent with each other but also coupled together. **The planning and execution depend on analysis of Knowledge representation and reasoning.**

Approaches to knowledge representation:

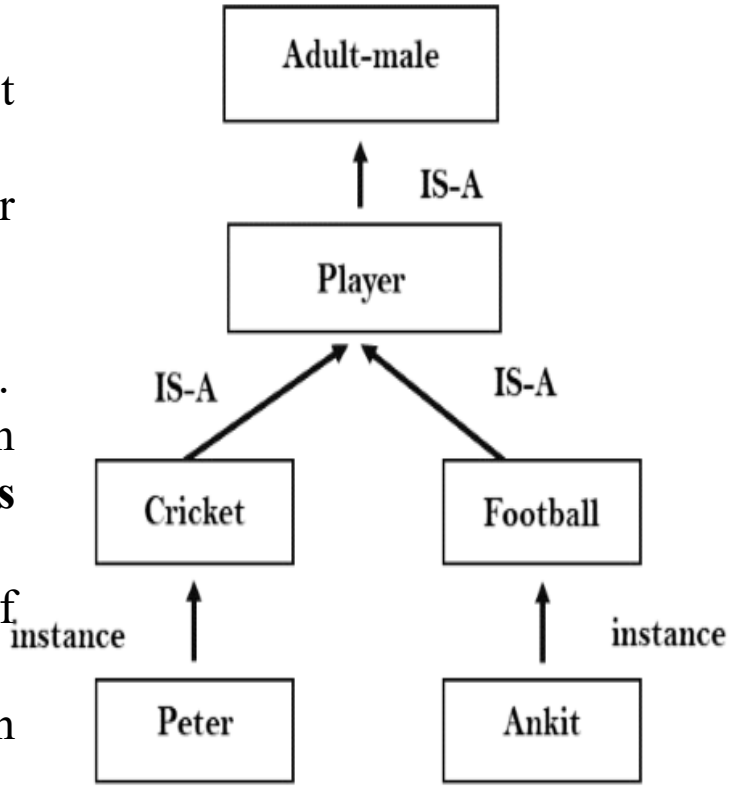
1. Simple relational knowledge:

Player	Weight	Age
Player1	65	23
Player2	58	18
Player3	75	24

- It is the simplest way of storing facts which uses the relational method, and each fact about a set of the object is set out systematically in columns.
- This approach of knowledge representation is famous in database systems where the relationship between different entities is represented.
- This approach has little opportunity for inference.

2. Inheritable knowledge

- In the inheritable knowledge approach, all data must be stored into a hierarchy of classes.
- All classes should be arranged in a generalized form or a hierarchal manner.
- In this approach, we apply **inheritance property.**
- Elements inherit values from other members of a class.
- This approach contains inheritable knowledge which shows a relation between **instance and class, and it is called instance relation.**
- Every individual frame can represent the collection of **attributes and its value.**
- In this approach, objects and values are represented in Boxed nodes.



3. Inferential knowledge

- Inferential knowledge approach represents knowledge in the form of formal logics.
- This approach can be used to derive more facts.
- It guaranteed correctness.
- **Example:** Let's suppose there are two statements:
    a. Marcus is a man
    b. All men are mortal
    Then it can represent as;

    man(Marcus)
    ∀x = man (x) ----------> mortal (x)s

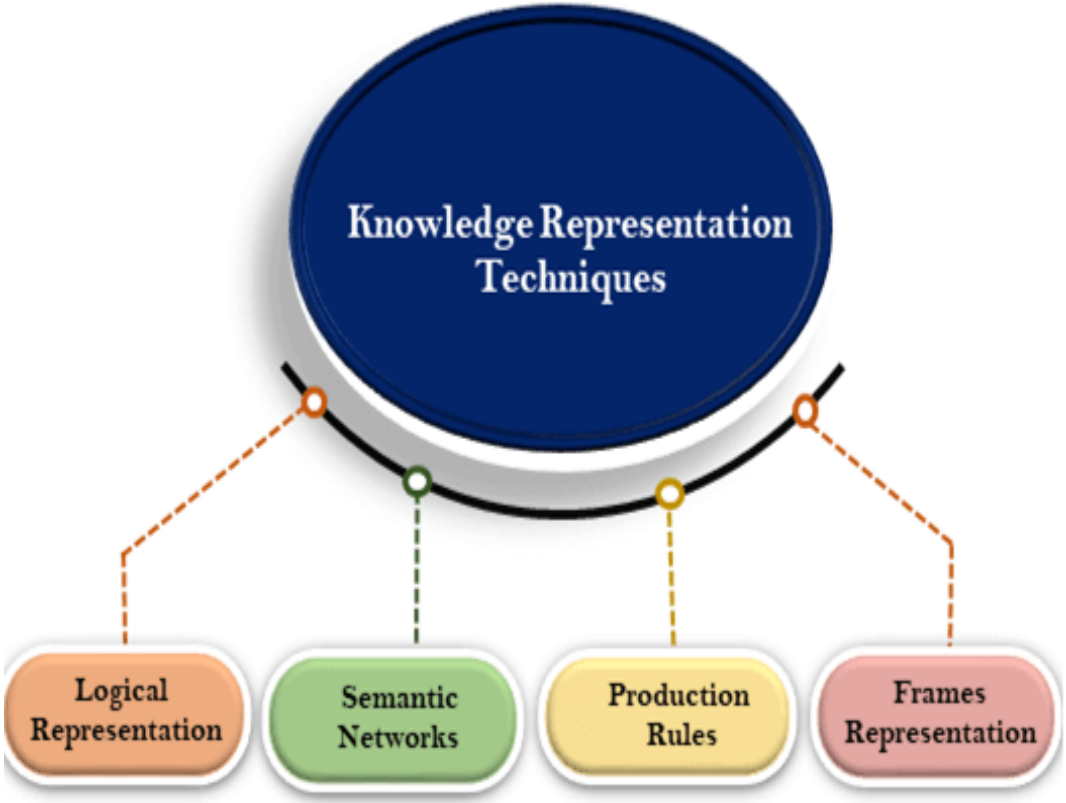
4. Procedural knowledge

- Procedural knowledge approach uses small programs and codes which describes how to do specific things, and how to proceed.
- In this approach, one important rule is used which is **If-Then rule**.
- In this knowledge, we can use various coding languages such as **LISP language** and **Prolog language**.
- We can easily represent heuristic or domain-specific knowledge using this approach.
- But it is not necessary that we can represent all cases in this approach.

Techniques of knowledge representation

There are mainly four ways of knowledge representation which are given as follows:

- Logical Representation
- Semantic Network Representation
- Frame Representation
- Production Rules



1. Logical representation

Logical representation is a language with some concrete rules which deals with propositions and has no ambiguity in representation. Logical representation means drawing a conclusion based on various conditions. It consists of precisely defined syntax and semantics which supports the sound inference. Each sentence can be translated into logics using syntax and semantics.

Syntax:

Syntaxes are the rules which decide how we can construct legal sentences in the logic.

It determines which symbol we can use in knowledge representation.

How to write those symbols.

Semantics:

Semantics are the rules by which we can interpret the sentence in the logic.

Semantic also involves assigning a meaning to each sentence.

Logical representation can be categorised into mainly two logics:

- **Propositional Logics**
- **Predicate logics**

Advantages of logical representation:

1. Logical representation enables us to do logical reasoning.
2. Logical representation is the basis for the programming languages.

Disadvantages of logical Representation:

1. Logical representations have some restrictions and are challenging to work with.
2. Logical representation technique may not be very natural, and inference may not be so efficient.

2. Semantic Network Representation

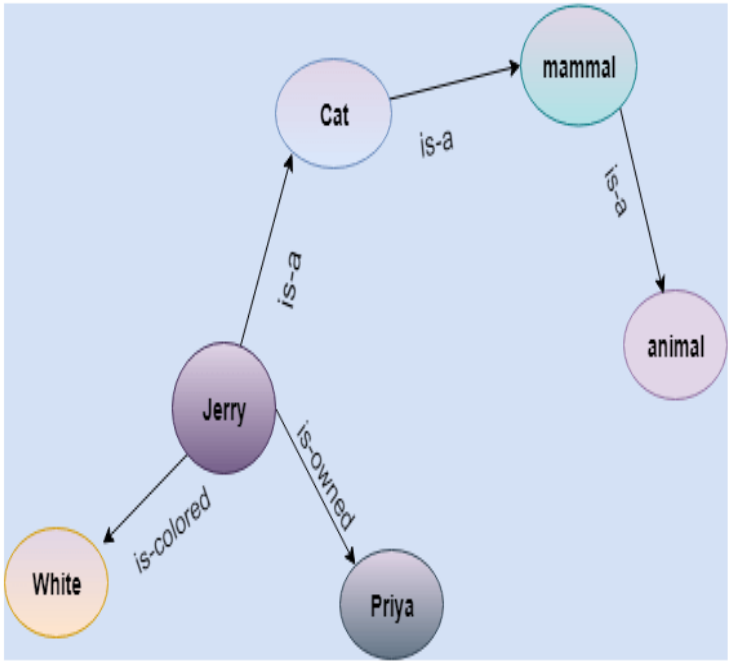
- Setworks are alternative of predicate logic for knowledge representation.
- In Semantic networks, we can represent our knowledge in the **form of graphical networks.**
- This network consists of nodes representing objects and arcs which describe the relationship between those objects.
- Semantic networks can categorize the object in different forms and can also link those objects.

This representation consist of mainly two types of relations:

- IS-A relation (Inheritance)
- Kind-of-relation

Example: Following are some statements which we need to represent in the form of nodes and arcs.

Statements:
Jerry is a cat.
Jerry is a mammal
Jerry is owned by Priya.
Jerry is brown colored.
All Mammals are animal.



Drawbacks in Semantic representation:

- Semantic networks take more computational time at runtime as we need to traverse the complete network tree to answer some questions. It might be possible in the worst case scenario that after traversing the entire tree, we find that the solution does not exist in this network.
- Semantic networks try to model human-like memory (Which has 1015 neurons and links) to store the information, but in practice, it is not possible to build such a vast semantic network.
- These types of representations are inadequate as they do not have any equivalent quantifier, e.g., for all, for some, none, etc.
- Semantic networks do not have any standard definition for the link names.

Advantages of Semantic network:

- Semantic networks are a natural representation of knowledge.
- Semantic networks convey meaning in a transparent manner.
- These networks are simple and easily understandable.

3. Frame Representation

A frame is a record like structure which consists of a collection of attributes and its values to describe an entity in the world.

Frames are the AI data structure which **divides knowledge into substructures by representing stereotypes situations.**

It consists of a collection of slots and slot values. These slots may be of any type and sizes. Slots have names and values which are called **facets.**

Slots	Filters
Title	Artificial Intelligence
Genre	Computer Science
Author	Peter Norvig
Edition	Third Edition
Year	1996
Page	1152

The above example was of a frame for a book

let us consdier one more example ....

Example 2:

Let's suppose we are taking an entity, Peter. Peter is an engineer as a profession, and his age is 25, he lives in city London, and the country is England. So following is the frame representation for this:

Slots	Filter
Name	Peter
Profession	Doctor
Age	25
Marital status	Single
Weight	78

Advantages of frame representation:

1. The frame knowledge representation makes the programming easier by grouping the related data.
2. The frame representation is comparably flexible and used by many applications in AI.
3. It is very easy to add slots for new attribute and relations.
4. It is easy to include default data and to search for missing values.
5. Frame representation is easy to understand and visualize.

Disadvantages of frame representation:

1. In frame system inference mechanism is not be easily processed.
2. Inference mechanism cannot be smoothly proceeded by frame representation.
3. Frame representation has a much generalized approach.

4. Production Rules

Production rules system consist of (**condition, action**) pairs which mean, "If condition then action". It has mainly three parts:

- The set of production rules
- Working Memory
- The recognize-act-cycle

In production rules agent checks for the condition and if the condition exists then production rule fires and corresponding action is carried out.
The condition part of the rule determines which rule may be applied to a problem. And the action part carries out the associated problem-solving steps.
This complete process is called a **recognize-act cycle.**

Example:

- IF (at bus stop AND bus arrives) THEN action (get into the bus)
- IF (on the bus AND paid AND empty seat) THEN action (sit down).
- IF (on bus AND unpaid) THEN action (pay charges).
- IF (bus arrives at destination) THEN action (get down from the bus).

Advantages of Production rule:

1. The production rules are expressed in natural language.
2. The production rules are highly modular, so we can easily remove, add or modify an individual rule.

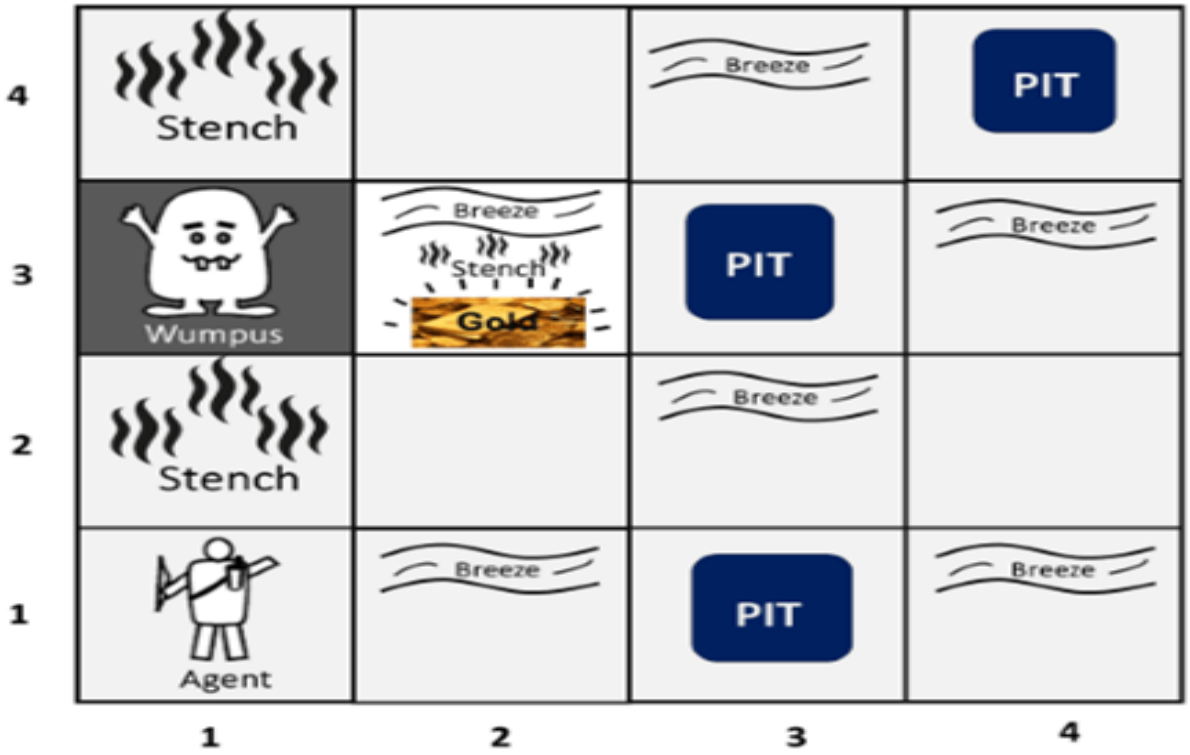
Disadvantages of Production rule:

1. Production rule system does not exhibit any learning capabilities, as it does not store the result of the problem for the future uses.
2. During the execution of the program, many rules may be active hence rule-based production systems are inefficient.

WAMPUS WORLD PROBLEM

- 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.
- 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.

- Following is a sample diagram for representing the Wumpus world. It is shov m with quare locat



There are also some components which can help the agent to navigate the cave. These components are given as follows:

1. The rooms adjacent to the Wumpus room are smelly, so that it would have some stench.
2. The room adjacent to PITs has a breeze, so if the agent reaches near to PIT, then he will perceive the breeze.
3. There will be glitter in the room if and only if the room has gold.
4. 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:

- To explain the Wumpus world we have given PEAS description as below:

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.

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.

Sensors:

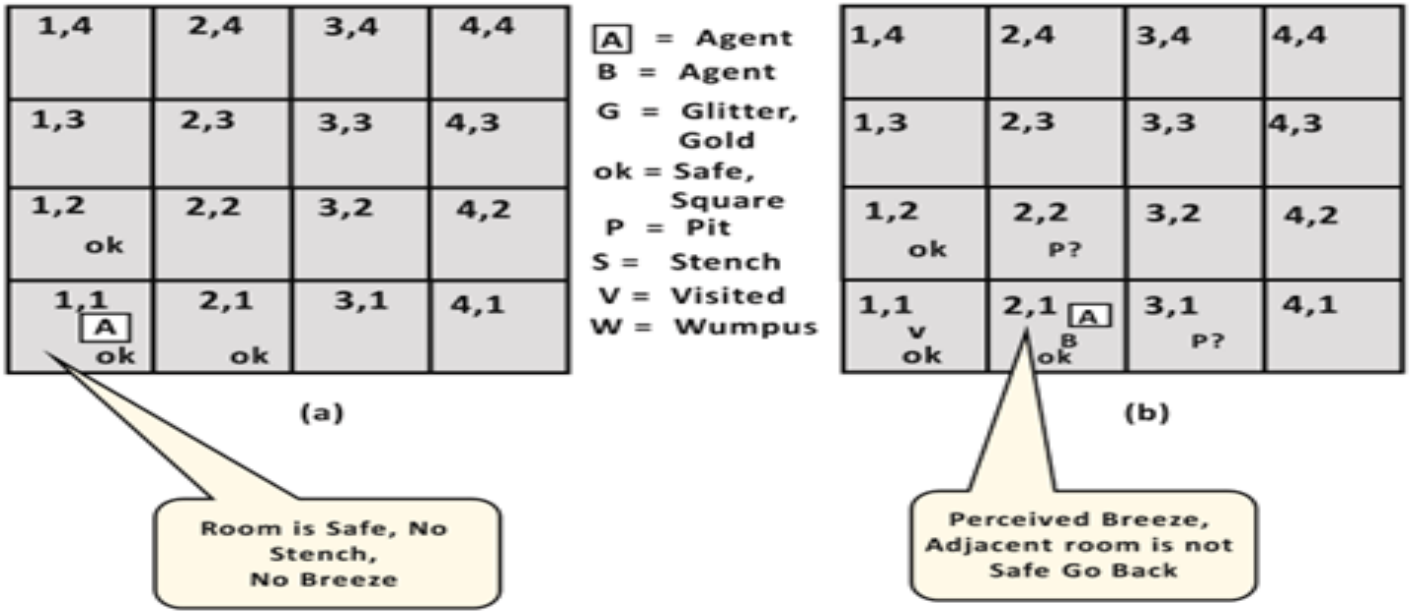
- The agent will perceive the stench if he is in the room adjacent to the Wumpus. (Not diagonally).
- The agent will perceive breeze if he is in the room directly adjacent to the Pit.
- The agent will perceive the glitter in the room where the gold is present.
- The agent will perceive the bump if he 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.

Exploring the Wumpus world:

Now we will explore the Wumpus world and will determine how the agent will find its goal by applying logical reasoning.

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 us e, G for its, W for
- At any Ste so OK.



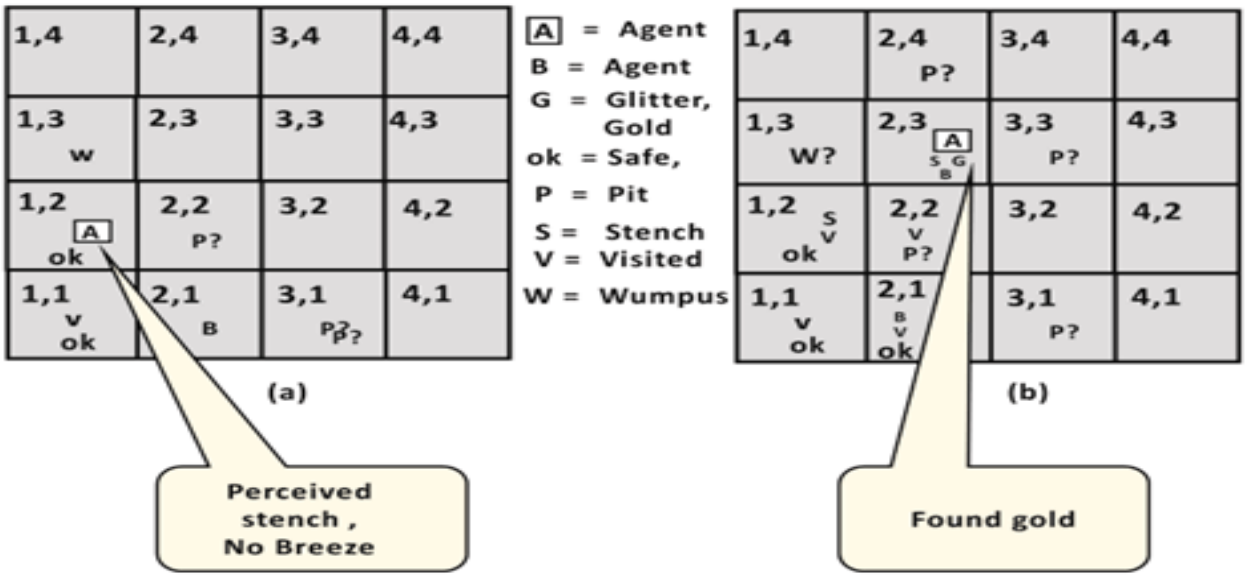
Agent's second Step:

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.

Agent's third 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].



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 (PL)

Propositional logic (PL) is the simplest form of logic where all the statements are made by propositions. A proposition is a declarative statement which is either true or false. It is a technique of knowledge representation in logical and mathematical form.

- **Example**:
- a) It is Sunday.
- b) The Sun rises from West (False proposition)
- c) 3+3= 7(False proposition)
- d) 5 is a prime number.

- **Following are some basic facts about propositional logic:**

✓ Propositional logic is also called Boolean logic as it works on 0 and 1.

✓ In propositional logic, we use symbolic variables to represent the logic, and we can use any symbol for a representing a proposition, such A, B, C, P, Q, R, etc.

✓ Propositions can be either true or false, but it cannot be both.

- ✓ The propositions and connectives are the basic elements of the propositional logic.
- ✓ Connectives can be said as a logical operator which connects two sentences.
- ✓ A proposition formula which is always true is called **tautology**, and it is also called a valid sentence.
- ✓ A proposition formula which is always false is called **Contradiction**.
- ✓ A proposition formula which has both true and false values is called **Contingency.**

- Statements which are questions, commands, or opinions are not propositions such as "**Where is Rohini**", "**How are you**", "**What**

Syntax of propositional logic:

- The syntax of propositional logic defines the allowable sentences for the knowledge representation. There are two types of Propositions:
- **Atomic Propositions**
- **Compound propositions**

1. Atomic Proposition: Atomic propositions are the simple propositions. It consists of a single proposition symbol. These are the sentences which must be either true or false.

- **Example:**
- a) 2+2 is 4, it is an atomic proposition as it is a **true** fact.
- b) "The Sun is cold" is also a proposition as it is a **false** fact.

2.Compound proposition: Compound propositions are constructed by combining simpler or atomic propositions, using parenthesis and logical connectives.

Example:

- a) "It is raining today, and street is wet."
- b) "Ankit is a doctor, and his clinic is in Mumbai."

Logical Connectives:

- Logical connectives are used to connect **two simpler propositions or representing a sentence logically**. We can create compound propositions with the help of logical connectives.

There are mainly ***five connectives,*** which are given as follows:

- **1.Negation:** A sentence such as ¬ **P** is called negation of P. A literal can be either Positive literal or negative literal.
- **2. Conjunction:** A sentence which has ∧ connective such as, **P ∧ Q** is called a conjunction.

Example: Rohan is intelligent and hardworking. It can be written as,
P= Rohan is intelligent,
Q= Rohan is hardworking. → P∧ Q.
•

- **3. Disjunction:** A sentence which has ∨ connective, such as **P ∨ Q**. is called disjunction, where P and Q are the propositions.
  Example: "Ritika is a doctor or Engineer",

Here P= Ritika is Doctor. Q= Ritika is Doctor, so we can write it as **P ∨ Q**.

- **4. Implication:** A sentence such as P → Q, is called an implication. Implications are also known as if-then rules. It can be represented as
  Example: If it is raining, then the street is wet.
  Let P= It is raining, and Q= Street is wet, so it is represented as P → Q

- **5. Biconditional:** A sentence such as **P⇔ Q is a Biconditional sentence, example If I am breathing, then I am alive**
  P= I am breathing, Q= I am alive, it can be represented as P ⇔ Q.

Table for Propositional Logic Connectives:

Connective symbols	Word	Technical term	Example
∧	AND	Conjunction	A ∧ B
∨	OR	Disjunction	A ∨ B
→	Implies	Implication	A → B
⇔	If and only if	Biconditional	A⇔ B
¬ or ~	Not	Negation	¬ A or ¬ B

Truth Table:

In propositional logic, we need to know the truth values of propositions in all possible scenarios. We can combine all the possible combination with logical connectives, and the representation of these combinations in a tabular format is called **Truth table**. Following are the truth table for all logical connectives:

For Negation:

P	¬ P
True	**False**
False	**True**

For Conjunction:

P	Q	P∧ Q
True	True	**True**
True	False	**False**
False	True	**False**
False	False	**False**

For disjunction:

P	Q	P ∨ Q.
True	True	**True**
False	True	**True**
True	False	**True**
False	False	**False**

For Implication:

P	Q	P→ Q
True	True	**True**
True	False	**False**
False	True	**True**
False	False	**True**

For Biconditional:

P	Q	P⇔ Q
True	True	**True**
True	False	**False**
False	True	**False**
False	False	**True**

- **<u>Limitations of Propositional logic:</u>**
- We cannot represent relations like ALL, some, or none with propositional logic. Example:
  - **All the boys are intelligent.**
  - **Some apples are sweet.**
- Propositional logic has limited expressive power.
- In propositional logic, we cannot describe statements in terms of their properties or logical relationships.

Knowledge representation using predicate logic

- 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.

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

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

- First-order logic is another way of knowledge representation in artificial intelligence. It is an extension to propositional logic.

▪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:

▪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.

Basic Elements of First-order logic:

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

Atomic sentences vs. 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:

- 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 as a predicate.

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:

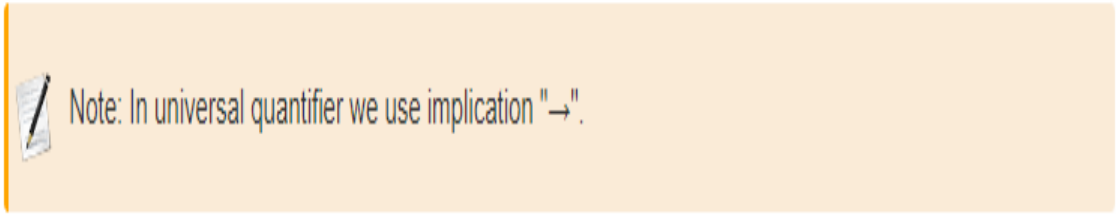
Universal Quantifier, (for all, everyone, everything)

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:

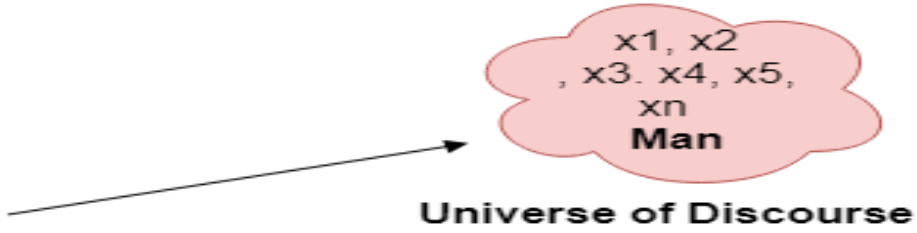
For all x
For each x
For every x.

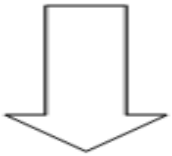
Example:

All man drink coffee.

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

- **x1 drinks coffee**
  ^
- **x2 drinks**
  ^
- **x3 drinks milk**
  ^
- .
- .
  ^
- **xn drinks milk**





So in shorthand notation, we can write it as :

∀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.

- Note: In Existential quantifier we always use AND or Conjunction symbol (∧).

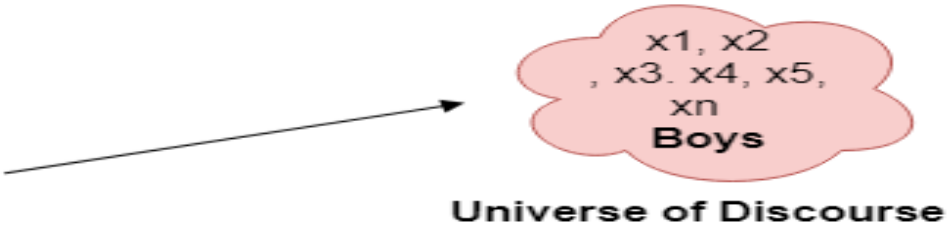
If x is a variable, then existential quantifier will be ∃x or ∃(x). And it will be read as:

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

Example:

Some boys are intelligent.

- **x1 is intelligent**
  V
- **x2 is intelligentV**
- **x3 is intelligent**
  V
- .
- .
  V
- **xn is intelligent**



Universe of Discourse

So in short-hand notation, we can write it as:

∃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.

Few examples

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.

$$\forall x \; bird(x) \rightarrow fly(x).$$

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:

$$\forall x \; man(x) \rightarrow respects\ (x,\ parent).$$

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**:

$$\exists x \; boys(x) \rightarrow play(x,\ cricket).$$

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:

$$\neg \forall (x)\ [\ student(x) \rightarrow like(x,\ Mathematics) \wedge like(x,\ Science)].$$

Knowledge Engineering in First-order logic(Predicate logic)

What is knowledge-engineering?

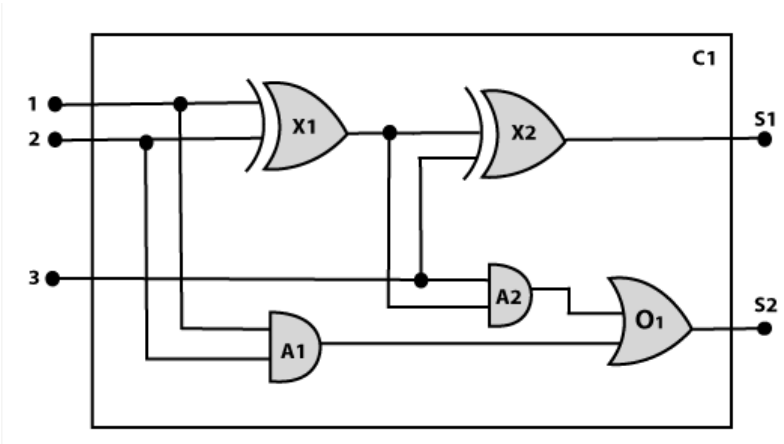
- The process of constructing a knowledge-base in first-order logic is called as **knowledge-engineering.**

- **Who is a Knowledge engineer?**

someone who investigates a particular domain, learns important concept of that domain, and generates a formal representation of the objects

The knowledge-engineering process

We will develop a knowledge base which will allow us to reason about digital circuit (**One-bit full adder**) which is given below.



A full adder circuit is central to most digital circuits that perform addition or subtraction. It is so called because it adds together two binary digits, plus a carry-in digit to produce a sum and carry-out digit. It therefore has three inputs and two outputs.

Basic Steps of Knowledge Engineering

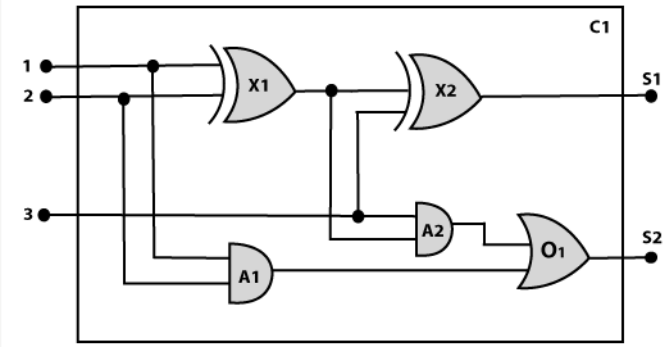
1. Identify the task .
2. Assemble the relevant knowledge.
3. Decide on vocabulary
4. Encode general knowledge about the domain.
5. Encode a description of the problem instance.
6. Pose queries to the inference procedure and get answers.
7. Debug the knowledge base

For the digital circuit, there are various reasoning tasks!

Examining the functionality of the circuit:

Does the circuit add properly?

What will be the output of gate A2, if all the inputs are high?



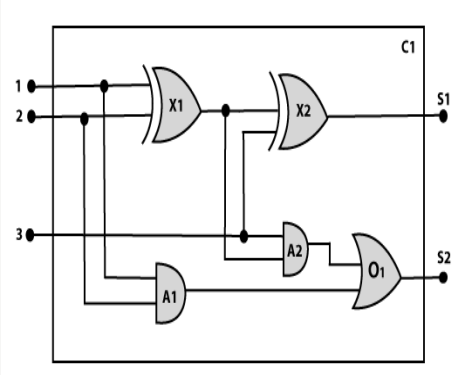
Examine the circuit structure details such as:

Which gate is connected to the first input terminal?

Does the circuit have feedback loops?

In the second step, we will assemble the relevant knowledge which is required for digital circuits.

So for digital circuits, we have the following required knowledge:

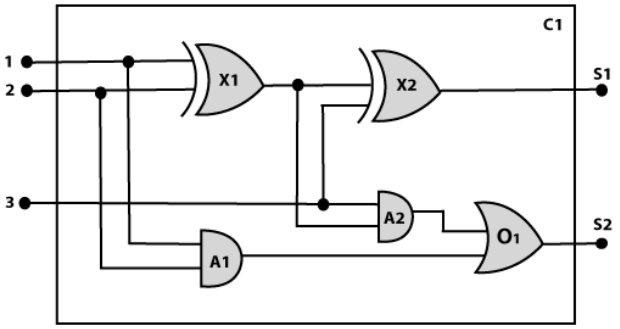


- Logic circuits are made up of wires and gates.
- Signal flows through wires to the input terminal of the gate, and each gate produces the corresponding output which flows further.
- In this logic circuit, there are four types of gates used: **AND, OR, XOR, and NOT**.
- All these gates have one output terminal and two input terminals (except NOT gate, it has one input terminal).

Process is to select functions, predicate, and constants to represent the circuits, terminals, signals, and gates.

- Each gate is represented as an object which is named by a constant, such as, **Gate(X1)**.

- The functionality of each gate is determined by its type, which is taken as as **AND, OR, XOR, or NOT**.



- Circuits will be identified by a predicate: **Circuit (C1)**.

- For the terminal, we will use predicate: **Terminal(x)**.

- For gate input, we will use the function **In(1, X1)** for denoting the first input terminal of the gate, and for output terminal we will use **Out (1, X1)**.

- The function **Arity(c, i, j)** is used to denote that circuit c has i input, j output.

- The connectivity between gates can be represented by predicate **Connect(Out(1, X1), In(1, X1))**.

- We use a unary predicate **On (t)**, which is true if the signal at a terminal is on.

To encode the general knowledge about the logic circuit, we need some following rules:

- If two terminals are connected then they have the same input signal, it can be represented as:

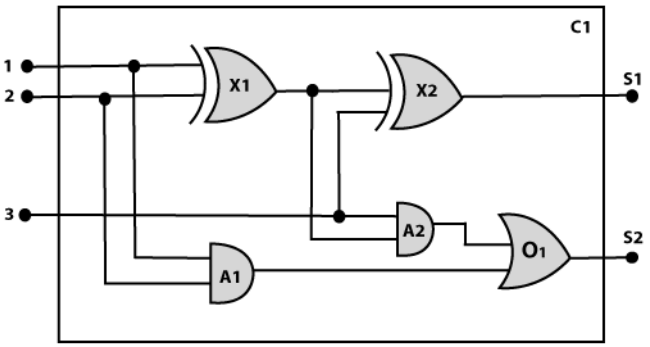
$$\forall\ t1, t2\ \text{Terminal}(t1) \wedge \text{Terminal}(t2) \wedge \text{Connect}(t1, t2) \rightarrow \text{Signal}(t1) = \text{Signal}(t2).$$

- Signal at every terminal will have either value 0 or 1, it will be represented as:

$$\forall\ t\ \text{Terminal}(t) \rightarrow \text{Signal}(t) = 1 \vee \text{Signal}(t) = 0.$$

- Connect predicates are commutative:

$$\forall\ t1, t2\ \text{Connect}(t1, t2) \rightarrow \text{Connect}(t2, t1).$$



▪Representation of types of gates:

$\forall$ g Gate(g) $\wedge$ r = Type(g) $\rightarrow$ r = OR $\vee$r = AND $\vee$r = XOR $\vee$r = NOT.

▪Representation of types of gates:
$\forall$ g Gate(g) $\wedge$ r = Type(g) $\rightarrow$ r = OR $\vee$r = AND $\vee$r = XOR $\vee$r = NOT.

▪Output of AND gate will be zero if and only if any of its input is zero.

$\forall$ g Gate(g) $\wedge$ Type(g) = AND $\rightarrow$Signal (Out(1, g))= 0 $\Leftrightarrow$ $\exists$n Signal (In(n, g))= 0.

▪Output of OR gate is 1 if and only if any of its input is 1

$\forall$ g Gate(g) $\wedge$ Type(g) = OR $\rightarrow$ Signal (Out(1, g))= 1 $\Leftrightarrow$ $\exists$n Signal (In(n, g))= 1

- Output of XOR gate is 1 if and only if its inputs are different:

$\forall$ g Gate(g) $\wedge$ Type(g) = XOR $\rightarrow$ Signal (Out(1, g)) = 1 $\Leftrightarrow$ Signal (In(1, g)) $\neq$ Signal (In(2, g))

- Output of NOT gate is invert of its input:

$\forall$ g Gate(g) $\wedge$ Type(g) = NOT $\rightarrow$ Signal (In(1, g)) $\neq$ Signal (Out(1, g)).

- All the gates in the above circuit have two inputs and one output (except NOT gate)

$\forall$ g Gate(g) $\wedge$ Type(g) = NOT $\rightarrow$ Arity(g, 1, 1)
$\forall$ g Gate(g) $\wedge$ r =Type(g) $\wedge$ (r= AND $\vee$r= OR $\vee$r= XOR) $\rightarrow$ Arity (g, 2, 1).

- All gates are logic circuits

$\forall$ g Gate(g) $\rightarrow$ Circuit (g).

5. Encode a description of the problem instance

- In the circuit there are two XOR, two AND and one OR gate.

atomic sentences for these gates are :

For XOR gate: Type(x1)= XOR, Type(X2) = XOR
For AND gate: Type(A1) = AND, Type(A2)= AND
For OR gate: Type (O1) = OR.

- In this step, we will find all the possible set of values of all the terminal for the adder circuit.

- The first query will be:

What should be the combination of input which would generate the first output of circuit C1, as 0 and a second output to be 1?

$\exists$ i1, i2, i3 Signal (In(1, C1))=i1 $\wedge$ Signal (In(2, C1))=i2 $\wedge$ Signal (In(3, C1))= i3
$\wedge$ Signal (Out(1, C1)) =0 $\wedge$ Signal (Out(2, C1))=1

7. Debug the knowledge base:

- Now we will debug the knowledge base, and this is the last step of the complete process. In this step, we will try to debug the issues of knowledge base.

In the knowledge base, we may have omitted assertions like $1 \neq 0$

END OF MODULE 2