**Unit-2**

**Q.1) Explain The Predicate Logic In Brief?**

* It is another way of knowledge representation in artificial intelligence
* Known as FOPL (First order predicate logic) and FOL (First order logic)
* Extension of propositional logic (connectives: Λ, ⋁, ~ , →, ↔)
* **Example:** A is a Brother of B

Brother\_of (A,B)

* What FOL will follow to represent the things in the real world?

**1.** **Object:** People, numbers, colors

**2. Constant:** a proper noun (name of a person, place)

**3. Variable:** x, y, a, b, c…….etc

**4.** **Function:** F\_name(------)

**Example:** A is a brother of B brother\_of (A,B)

* The part of a sentence with the verb tells us what the subject is or does.
* In the sentence—‘He went cycling after returning from school’ the predicate is ‘went cycling after returning from school’
* Relation between two objects.
* **FOL:** A very basic sentence is called an atomic sentence.
* These sentences are formed in FOL by a predicate symbol followed by some parenthesis with the sequence of terms (subjects).
* **Syntax :** Predicate (term1,…….term**n** )

1. X is a brother of Y

brother\_of (X,Y)

brother(X,Y)

1. X is an integer

is\_an\_integer (X) integer (X)

1. Dove is a bird

Is\_a\_bird (Dove)

Bird (Dove)

* **Quantifiers:**
* Defines the scope of the object.
* Types of Quantifier:

1. Universal quantifier (∀)

2. Existential quantifier (∃)

* **(∀)X :** - For each x

- For all x

- For every x

* **(∃) X :** - There exists x

- For some x

- For at least one x

* **Examples:**
* Some girls are intelligent.

∃ : girl(X) Λ intelligent (X)

* All men drink tea

∀ 𝑥: man(X)→drink (X,tea)

* Some employs are sick.

∃ : employ(X) Λ sick (X)

* Every man respects his parent

∀ 𝑥: man(X)→respect (X, Parent)

* Ravi is a brother of Ram

Brother\_of (Ravi,Ram)

* Some boys play cricket

∃ : boys(X) Λ Play (X,cricket)

* Every student takes at least one course.

∀ 𝑥: students(X)→ ∃𝑦 (course(y) Λ takes(x,y))

* Every student who takes Analysis also takes Geometry

∀ : students(X) Λ takes X, Analysis −→ tkaes (X,Geometry)

**Q.2) White A Short Note On The Instance And isa relationship?**

* Specific attributes instance and isa play an important role, particularly in a useful form of reasoning called property inheritance.
* The predicates instance and explicitly captured the relationships they used to express, namely class membership and class inclusion.

**Instance:** Indicates class membership

**Isa:** indicates class inclusion

* Three ways of representing class membership
* **First way (Representation):**

1. Man(Marcus)
2. Pompeian (Marcus)
3. ∀ 𝑥: Pompeian(x) → Roman(x)
4. Ruler (Caesar)
5. ∀ 𝑥: Romans(x) → loyalto(x,Ceasar) V hate(x,Ceasar)

* **Second way (Use of instance predicate)**

1. Instance (Marcus, man)
2. Instance (Marcus, Pompeian)
3. ∀ 𝑥: Instance(x, Pompeian) → Instance(x, Roman)
4. Instance (Caesar, Ruler)
5. ∀ 𝑥: instance(x,Roman) → loyalto(x,Ceasar) V hate(x,Ceasar)

* **Third way** (use both instance and isa predicate)

1. Instance (Marcus, man)
2. Instance (Marcus, Pompeian)
3. Isa(Pompeian, Roman)
4. Instance (Caesar, Ruler)
5. ∀ 𝑥: instance(x,Roman) → loyalto(x,Ceasar) V hate(x,Ceasar)
6. ∀ 𝑥∀ y∀ z Instance(x,y) V isa(y,z) →

**Q.3) Explain in brief Computable functions and predicates?**

* It includes simple facts such as greater than (gt) or less than (lt) relationships

gt(1,0) lt(0,1)

gt(2,1) lt(1,2)

gt(3,2) lt(2,3)

* gt(2+3,1): we first compute the value of the plus function given the arguments 2 and 3 , then send the arguments 5 and 1 to gt.
* Consider the following sets of facts, again involving Marcus:

1. Marcus was a man

man(Marcus)

1. Marcus was a Pompeian.

Pompeian(Marcus)

1. Marcus was born in 40 A.D.

born(Marcus,40)

1. All men are mortal

∀ 𝑥: man(x)→Mortal(X)

1. All Pompeians died when Volcano erupted in 79A.D.

erupted(volcano,79) ∧∀ 𝑥 :[pompeians (X) →dide(x,79)]

1. No mortal lives longer than 150 years.

∀ 𝑥: ∀ 𝑡1: ∀ t2: mortal(x) ∧born(x,t1) ∧gt(t2-t1,150) ∧→ 𝑑𝑖𝑒𝑑(𝑥,𝑡2)

1. It is now 1991.

now = 1991 Now

* Suppose we want to answer the question “Is Marcus alive?”

**a)** Either we can show that Marcus is dead because he was killed by the volcano.

**b)** He must be dead because he would otherwise be more than 150 years old, which we know is not possible.

1. Alive means not dead.

∀ 𝑥: ∀ 𝑡:[𝑎𝑙𝑖𝑣𝑒 𝑥,𝑡 → ¬dead(x,t)] ∧¬]dead(x,t)] → 𝑎𝑙𝑖𝑣𝑒 𝑥,𝑡 ]

1. If someone dies, then he is dead at all later times.

∀ 𝑥: ∀ 𝑡1: ∀ t2: died(x,t1) ∧gt(t2,t1) → 𝑑𝑒𝑎𝑑(𝑥,𝑡2)

**Q.4) State and explain the resolution with an example?**

* **Steps for resolution:**

1. Arya likes all kinds of food
2. Apples and vegetables are food
3. Anything anyone eats and is not killed is food.
4. Ajay eats peanuts and is still alive

**Prove: Arya likes peanuts.**

1. **Negate the statement to be proved Prove:**

Arya likes peanuts.

likes(Arya, peanuts) ¬ 𝑙𝑖𝑘𝑒𝑠(𝐴𝑟𝑦𝑎, 𝑝𝑒𝑎𝑛𝑢𝑡𝑠)

1. **Conversion of all facts into first-order logic**

**1**. Arya likes all kinds of food.

∀ 𝑥:food(X)→likes (Arya,x)

**2.** Apples and vegetables are food.

Food(Apple) ∧ Food (vegtables)

**3.** Anything anyone eats and is not killed is food.

∀ 𝑥 ∀ 𝑦 ∶𝑒𝑎𝑡𝑠 (𝑥, 𝑦) ∧ ¬ killed (𝑥)→ f𝑜𝑜𝑑 (𝑦)

**4.** Ajay eats peanuts and still alive

Eats(Ajay, 𝑝𝑒𝑎𝑛𝑢𝑡𝑠) ∧ alive(Ajay)

1. **Convert the FOL statement into CNF :**

**1**. Eliminate ‘→’ and ‘↔’

𝑎 → 𝑏 = ¬𝑎⋁𝑏

a↔b=a→ b∧ b ↔ a

**2**. Move ‘¬’ inward:

¬(∀ 𝑥p) = ∃𝑥¬p

¬(∃𝑥𝑝) = ∀ 𝑥 ¬p

¬(a ⋁ b) = ¬a ∧ ¬b

¬(a ∧b) = ¬a ⋁ ¬𝑏

¬¬a = a

**3.** Rename variable

**4.** Replace existential quantifier with Skolem constant

∃𝑥 𝑅𝑖𝑐ℎ 𝑥 = 𝑅𝑖𝑐ℎ 𝐺1

**5.**Drop universal quantifier.

1. ∀ 𝑥:food(X)→likes (Arya,x)

¬Food(x) ⋁ likes (Arya,x)

2. ∀ 𝑥 ∀ 𝑦 ∶𝑒𝑎𝑡𝑠 (𝑥, 𝑦) ∧¬ killed (𝑥) → f𝑜𝑜𝑑 (𝑦)

¬[eats(x,y) ∧¬killed(x)]⋁ food (y)

¬ eats(x,y) ⋁ killed(x) ⋁ food(y).

1. **Resolution graph CNF(Conjuctive Norml Form):**
2. ¬Food(x) ⋁ likes (Arya,x)
3. Food(apple) ∧ food (vegetable)
4. ¬ eats(x,y) ⋁ killed(x) ⋁ food(y)
5. 𝑒𝑎𝑡𝑠 (Aj𝑎𝑦, 𝑝𝑒𝑎𝑛𝑢𝑡𝑠) 𝑎𝑛𝑑 𝑎𝑙𝑖𝑣𝑒 (𝑎𝑗𝑎𝑦)
6. 𝑘𝑖𝑙𝑙𝑒𝑑 𝑥 𝑉 𝑎𝑙𝑖𝑣𝑒 𝑥
7. ¬ alive x V ¬killed(x)

**Prove:** Arya likes peanuts. ¬ 𝑙𝑖𝑘𝑒𝑠(𝐴𝑟𝑦𝑎, 𝑝𝑒𝑎𝑛𝑢𝑡𝑠)

**Q.5) Give the diference between Procedural and Declarative knowledge?**

|  |  |
| --- | --- |
| **Procedural knowledge** | **Declarative knowledge** |
| Procedural knowledge means how a particular thing can be accomplished | Declarative knowledge means basic knowledge about something |
| It is also known as interpretive knowledge | It is also known as descriptive knowledge |
| Not more popular | More popular |
| Procedural knowledge can’t be easily communicate | Declarative knowledge can be easily communicate |
| Procedural knowledge is generally process-oriented in nature | Declarative knowledge is data oriented in nature |
| In Procedural knowledge debugging and validation is not easy | In Declarative knowledge debugging and validation is easy |
| Procedural knowledge is less effective in competitive programming | Declarative knowledge is more effective in competitive programming |

**Q.6) Compare forward chaining and backward chaining?**

|  |  |
| --- | --- |
| **Forward chaining** | **Backward chaining** |
| Starts with the initial facts. | Starts with some hypothesis or goal. |
| Asks many questions. | Asks few questions. |
| Tests all the rules. | Tests some rules. |
| Slow, because it tests all the rules. | Fast, because it tests fewer rules. |
| Provides a huge amount of information from just a small amount of data. | Provides a small amount of information from just a small amount of data |
| Attempts to infer everything possible from the available information. | Searches only that part of the knowledge base that is relevant to the current problem |
| Primarily data-driven | . Goal-driven |
| Uses input; searches rules for answer Attempts to infer everything possible from the available information. | Begins with a hypothesis; seeks information until the hypothesis is accepted or rejected. |
| Top-down reasoning | Bottom-up reasoning |
| Works forward to find conclusions from facts | Works backward to find facts that support the hypothesis |
| Tends to be breadth-first | Tends to be depth-first |
| Suitable for problems that start from data collection, e.g. planning, monitoring, control | Tends to be depth-first Suitable for problems that start from a hypothesis, e.g. diagnosis |
| Non-focused because it infers all conclusions, may answer unrelated questions | Focused; questions all focused to prove the goal and search as only the part of KB that is related to the problem |
| Explanation not facilitated | Explanation facilitated |
| All data is available | Data must be acquired interactively |
| A small number of initial states but a high number of conclusions | A small number of initial goals and a large number of rules match the facts |
| Forming a goal is difficult | Easy to form a goal |

**Q.7) Discuss the Logic Programming?**

* **PROLOG** (Programming in Logic)is one of the most widely used programming languages in artificial intelligence research.
* As opposed to imperative languages such as C or Java, it is a declarative programming language.
* In the prolog we declare some facts. These facts constitute the knowledge base of the system.
* We can query against the knowledge base.
* We get output as affirmative if our query is already in the knowledge base or it is implied by Knowledge Base, otherwise, we get output as negative.
* PROLOG facts are expressed in Horn clause (subset of FOL) or simply we can say in FOL
* Every fact ends with a dot (.)

**Example:** Friends (Sailee, Ashwini). : Sailee and Ashwini are friends

Singer (Sonu). : Sonu is a singer

Odd\_number(5). : 5 is an odd number

* **Query 1:** ?-Singer Sonu.

**Output:** Yes

**Explanation:** As our knowledge base contains the above fact, so the output was ‘Yes’, otherwise it would have been ‘No’

* **Query 1:** ?-odd\_number (7)

**Output:** No

**Explanation:** As our knowledge base does not contain the above fact, so the output was ‘No’

* **Horn clause:**
* A Horn Clause is a disjunction with at most one positive literal.

¬ 𝑥1 ∨ ¬x2 ∨ ¬x3 ∨……… ¬xn ∨ y

* Convert the following sentence in to Horn clause.

(𝑥1 ∧ 𝑥2 ∧ ⋯ ∧ 𝑥𝑛) → y

* Many facts (not all) can be translated into Horn Clause.

**For example**,

1. 𝐴 ∧ 𝐵 → 𝐶
2. 𝐴 ∨ 𝐵 → 𝐶
3. 𝐴 ∧ ¬𝐵 → 𝐶

**Q.8) Demonstrate the concept of knowledge engineering with suitable example?**

* The process of constructing a knowledge base in first-order logic is called as knowledge- engineering.
* In knowledge-engineering, someone who investigates a particular domain, learns important concept of that domain, and generates a formal representation of the objects, is known as knowledge engineer.
* **1. Identify the task:** At the first level, examine the functionality of the circuit:

1. Does the circuit add properly? 2. What will be the output of gate A2, if all the inputs are high?
2. At the second level, we will examine the circuit structure details:

1. Which gate is connected to the first input terminal?

2. Does the circuit have a feedback loop?

* **2. Assemble the relevant** knowledge 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 three types of gates used: AND, OR, XOR
* All these gates have one output terminal and two input terminals.
* **3. Decide on vocabulary:**  The next step of the process is to select predicates 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).
* Circuits will be identified by a predicate: Circuit (C1).
* For the terminal, we will use the 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 the 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))
* 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.
* **4. Encode the general knowledge about the domain:**

1. If two terminals are connected, then they have the same signal:

∀ 𝑡1,2: 𝑇𝑒𝑟𝑚𝑖𝑛𝑎𝑙 𝑡1 ∧ 𝑡𝑒𝑟𝑚𝑖𝑛𝑎𝑙 𝑡2 ∧ 𝑐𝑜𝑛𝑛𝑒𝑐𝑡𝑒𝑑 (𝑡1,𝑡2)

→ 𝑠𝑖𝑔𝑛𝑎𝑙 𝑡1 = 𝑠𝑖𝑔𝑛𝑎𝑙 (𝑡2)

1. Signal at every terminal is either 1 or 0:

∀ 𝑡: 𝑇𝑒𝑟𝑚𝑖𝑛𝑎𝑙 𝑡 → 𝑆𝑖𝑔𝑛𝑎𝑙 𝑡 = 1 ∨ 𝑠𝑖𝑔𝑛𝑎𝑙 𝑡 = 0

1. Connected is commutative

∀ 𝑡1,2: 𝑐𝑜𝑛𝑛𝑒𝑐𝑡𝑒𝑑 𝑡1,𝑡2 ↔ 𝑐𝑜𝑛𝑛𝑒𝑐𝑡𝑒𝑑 (𝑡2,𝑡1)

1. There are three types of gates:

∀ 𝑔: 𝐺𝑎𝑡𝑒 𝑔 ∧ 𝑘 = 𝑡𝑦𝑝𝑒 𝑔 → 𝑘 = 𝐴𝑁𝐷 ∨ 𝑘 = 𝑂𝑅 ∨ k = XOR

1. An AND’s gate output is 0 if and only if any of it’s input is 0:

∀ 𝑔 ∶ 𝐺𝑎𝑡𝑒 𝑔 ∧ 𝑡𝑦𝑝𝑒 𝑔 = 𝐴𝑁𝐷 → 𝑠𝑖𝑔(𝑜𝑢𝑡 1, 𝑔 = 0 ↔ ∋ 𝑛 𝑠𝑖𝑔𝑛𝑎𝑙 (𝐼𝑛( 𝑛, 𝑔)) =0

1. An OR gate’s output is 1 if and only if any of its input is 1 :

∀ 𝑔 ∶ 𝐺𝑎𝑡𝑒 𝑔 ∧ 𝑡𝑦𝑝𝑒 𝑔 = 𝑂𝑅 → 𝑠𝑖𝑔(𝑜𝑢𝑡 1, 𝑔 = 1 ↔ ∋ 𝑛 𝑠𝑖𝑔𝑛𝑎𝑙 𝐼𝑛 𝑛, 𝑔 = 1

1. An XOR gate’s output is 1 if and only if any of its inputs are different :

∀ 𝑔 ∶ 𝐺𝑎𝑡𝑒 𝑔 ∧ 𝑡𝑦𝑝𝑒 𝑔 = 𝑋𝑂𝑅 → 𝑠𝑖𝑔(𝑜𝑢𝑡 1, 𝑔 = 1 ↔ 𝑠𝑖𝑔𝑛𝑎𝑙 𝐼𝑛 1, 𝑔 ≠ 𝑠𝑖𝑔𝑛𝑎𝑙(𝐼𝑛(2, 𝑔 ))

1. A gates have two inputs and one output :

∀ 𝑔 ∶ 𝐺𝑎𝑡𝑒 𝑔 ∧ 𝑘 = 𝑡𝑦𝑝𝑒 𝑔 ∧ 𝑘 = 𝐴𝑁𝐷 ∨ 𝑘 = 𝑂𝑅 ∨ 𝑘 = 𝑋𝑂𝑅 → 𝐴𝑟𝑖𝑡𝑡𝑦 (𝑔, 2,1)

1. Gates are circuits. ∀ 𝑔 ∶

𝐺𝑎𝑡𝑒 𝑔 → 𝑐𝑖𝑟𝑐𝑢𝑖𝑡 (𝑔)

* **5. Encode a description of the problem:**  instance First , we categorize the circuit and its component gates:

𝑐𝑖𝑟𝑐𝑢𝑖𝑡 𝑐1 ∧ 𝐴𝑟𝑖𝑡𝑦 (𝑐1,3,2)

Gate 𝑥1 ∧ 𝑇𝑦𝑝𝑒 𝑥1 = 𝑋𝑂𝑅

Gate 𝑥2 ∧ 𝑇𝑦𝑝𝑒 𝑥2 = 𝑋𝑂𝑅

Gate 𝐴1 ∧ 𝑇𝑦𝑝𝑒 𝐴1 = 𝐴𝑁𝐷

Gate 𝐴2 ∧ 𝑇𝑦𝑝𝑒 𝐴2 = 𝐴𝑁𝐷

Gate 𝑂1 ∧ 𝑇𝑦𝑝𝑒 𝑂1 = 𝑂R

* **6. Pose queries to the inference procedure**

1. In this step, we will find all the possible sets of values of all the terminals for the adder circuit. The first query will be: What should be the combination of input that would generate the first output of circuit C1(sum bit), as 0, and a second output(carryout bit) to be 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 the knowledge base.
* In the knowledge base, we may have omitted assertions like 1 ≠ 0

**Q.9) Explain the Unification algorithm in detail?**

* Unification is a process of making two different logical atomic expressions identical by finding a substitution.
* Unification depends on the substitution method..
* **Example of unification:** 1. 𝑝(𝑥, 𝑓 𝑦 ) 𝑎𝑛𝑑 𝑝(𝑎, 𝑓(𝑔 𝑧 )

Sol: (𝑥, 𝑓 𝑦 (𝑎, (𝑔 𝑧 )

[a | x ]

𝑝(𝑎, 𝑓 𝑦 𝑝(𝑎, 𝑓(𝑔 𝑧 )

[ g(z) | y]

𝑝(𝑎, 𝑓 𝑔(𝑧))

**[a | x , g(z) | y]**

* **Conditions for Unification**: Predicate symbols must be the same, expressions with different predicate symbols can never be unified.
* Number of Arguments in both expressions must be identical.
* Unification will fail if there are two similar variables present in the same expression.
* **Unification Algorithm**:

Unify (L1,L2)

1. If L1 or L2 is a variable or constant, then
2. If L1 & L2 are identical return NIL
3. Else if L1 is a Variable,

Then if L1 occours in L2, then return FAILURE

Else return{(L2 /L1)}

1. Else if L2 is a Variable,

If L2 Occours in L1, then return FAILURE

Else return{(L1 /L2)}

1. Else return FAILURE
2. If the initialpredicate symbol and L2 are not same, then return FAILURE
3. If L1&L2 have a different numbers argument, then returnFAILURE
4. Set Substitution Set (SUBST) to NIL
5. For i=0 to the Number of Elements in L1
6. Call Unify Function with the ith element of L1 and element of L2 and put the result in S
7. If S = FAILURE, then return Failure
8. If S is not NIL then

Apple S to the remainder of both L1&L2

SUBST x Append (S,SUBST

1. Return SUBST

* **Implementation of Algorithm :**

1. Initialise the Substitutio set to be empty .A NIL set to be empty. A NIL Set indicate Failure
2. Recursively unify Expression

Check for identical expression match,

If one item is a variable Vi and other is term ti, not containing that variable then

1. Subtitute ti/vi in the existing Substitution
2. Add ti/vi to in the Substitution Set
3. If both item are functions, the function names must be identical and all arguments must be Same in both the Expression.

**11) What is Foward chaining?**

* When based on the available data a decision is taken then the process is Called as the Foward chaining.
* Its work from a initial Sate and reaches to Goal State.
* **Example:** It is a crime for an American to sell weapons to hostile nations.
* **American (p) ∧ weapon(q) ∧ sells (p, q, r) ∧ hostile(r) → Criminal(p)**

Country A has some missiles.

* **p Owns(A, p) ∧ Missile(p)**.All of the missiles were sold to country A by Robert.
* **p Missiles(p) ∧ Owns (A, p) → Sells (Robert, p, A)**Missiles are weapons.
* **Missile(p) → Weapons (p)**Enemy of America is known as hostile.
* **Enemy(p, America) →Hostile(p)** Country A is an enemy of America.
* **Enemy (A, America)** Robert is American
* **American(Robert).**

|  |
| --- |
|  |

**12) What is 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.
* **Example:**
* **American (p) ∧ weapon(q) ∧ sells (p, q, r) ∧ hostile(r) → Criminal(p)**   
  **Owns(A, T1)**
* **Missile(T1)**
* **?p Missiles(p) ∧ Owns (A, p) → Sells (Robert, p, A)**
* **Missile(p) → Weapons (p)**
* **Enemy(p, America) →Hostile(p)**
* **Enemy (A, America)**
* **American(Robert).**
* In Backward chaining, we will start with our goal predicate, which is **Criminal(Robert)**, and then infer further rules.

|  |
| --- |
|  |