UNIT-3
PART2

Dr. R. Seeta Sireesha
Associate Professor
CSE Department
GVPCE(A)

UNIT-3
PART2

Knowledge and Reasoning:

- Knowledge representation issues,
- predicate logic-Resolution, Unification,
- Representation knowledge using Rules-Inference in First – order logic forward and backward reasoning.

Knowledge and Reasoning

- ***Knowledge*** refers to the information that an AI system has about the world (or) Knowledge is the basic element for a human brain to know and understand the things logically.
- This information can be obtained through various means, such as sensors, databases, or human input.
- The knowledge can be in different forms, such as rules, facts, or heuristics.
- The more knowledge an AI system has, the better it can understand and reason about the world.
- ***Reasoning***, on the other hand, is the process by which an AI system uses its knowledge to draw conclusions or make decisions.

Knowledge and Reasoning (cont..)

- There are different types of reasoning, such as deductive reasoning, inductive reasoning, and abductive reasoning.
- In ***deductive reasoning***, an AI system applies logical rules to reach a conclusion based on given premises.
- In ***inductive reasoning***, an AI system derives general principles from specific observations.
- In ***abductive reasoning***, an AI system infers the best explanation for a given observation.

Example:

- To illustrate how knowledge and reasoning work in AI, imagine an AI system that helps ***diagnose medical conditions***.
- The system would need to have a large amount of medical knowledge, such as symptoms, treatments, and risk factors.
- It would then use deductive reasoning to match a patient's symptoms with known medical conditions and come up with a diagnosis.
- The system may also use inductive reasoning to learn from past cases and improve its accuracy over time.

What is Logic?

- Logic is the key behind any knowledge. It allows a person to filter the necessary information from the bulk and draw a conclusion.
- In AI, the representation of knowledge is done via logics. There are three main components of logic, which are as follows:
- **Syntax:** It is the sequence of a specific language which should be followed in order to form a sentence. Syntax is the representation of a language. Every language has its own syntax.

For example, ax_2+bx+c is a well-formed syntax of a quadratic equation.

What is Logic? (Cont..)

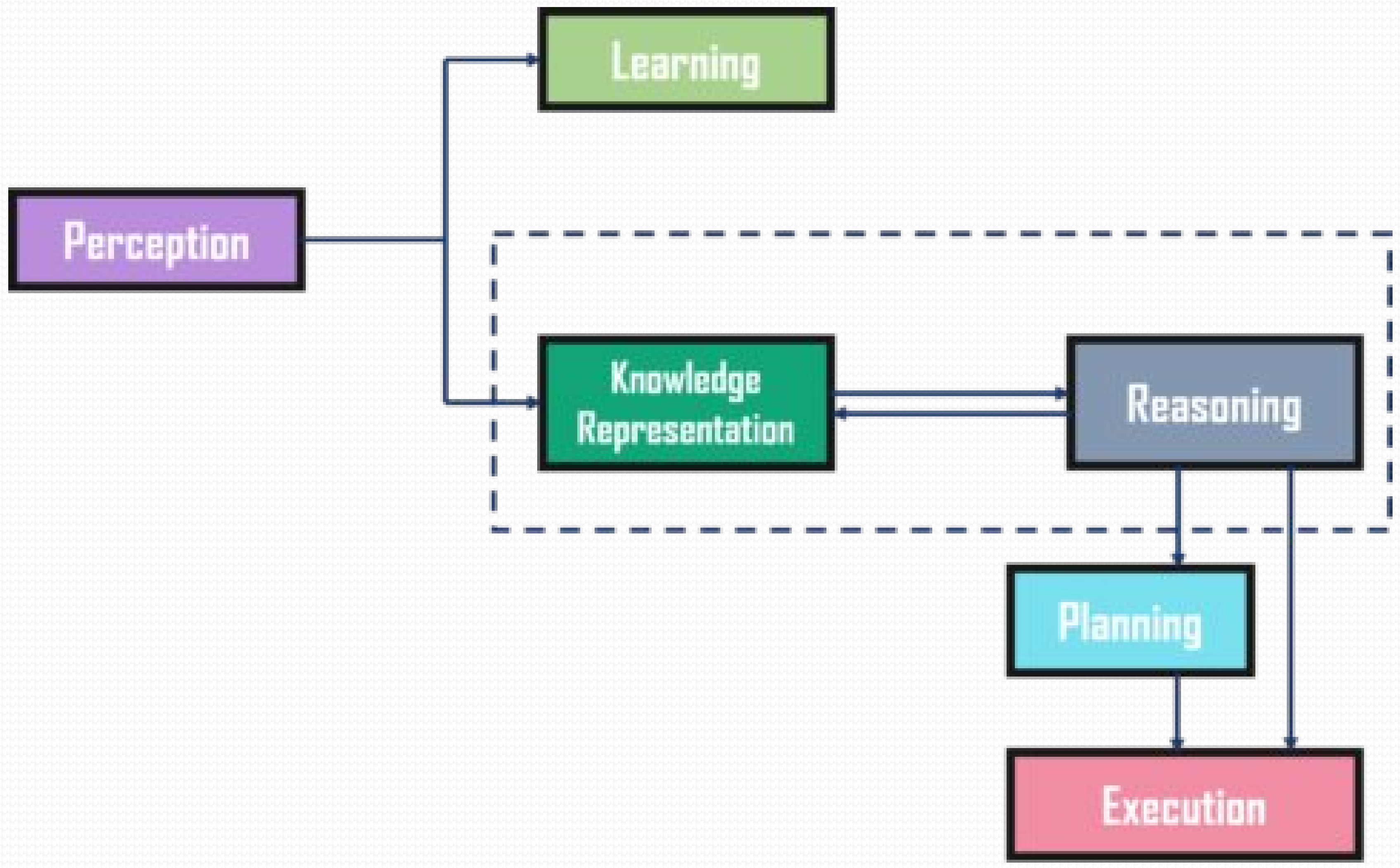
- **Semantics:** Semantics defines the sense of the sentence which relates to the real world.
  - **For example,** Indian people celebrate Diwali every year. This sentence represents the true fact about the country and its people who are Indians. Therefore, the sentence is syntactically as well as semantically correct.
- **Logical Inference:** Inference means to infer or draw some conclusions about some fact or a problem. Logical inference is thinking all the possible reasons which could lead to a proper result. Inference algorithms are used to perform logical inference.

Knowledge Representation

Knowledge-Based Agent: Knowledge-based agents are those agents who have the capability of **maintaining an internal state of knowledge, reason over that knowledge, update their knowledge after observations and take actions.**

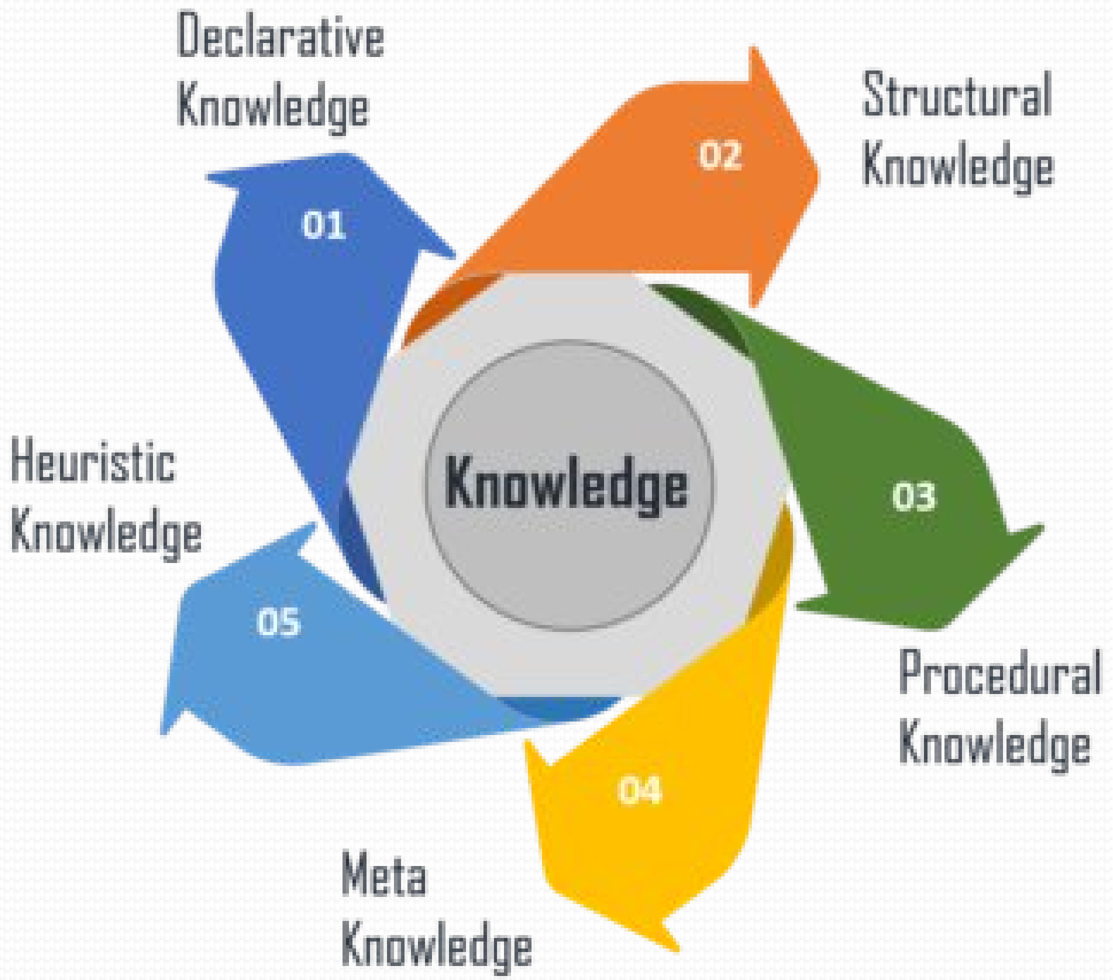
- Knowledge-based agents are composed of two main parts:
  - **Knowledge-base and**
  - **Inference system**.

Architecture of knowledge-based agent



- The **Perception component** retrieves data or information from the environment. Also, it defines how to respond when any sense has been detected.
- Then, there is the **Learning Component** that learns from the captured data by the perception component. In order to learn new things, the system requires knowledge acquisition, inference, acquisition of heuristics, faster searches, etc.
- The main component in the cycle is **Knowledge Representation and Reasoning** which shows the human-like intelligence in the machines. Knowledge representation is all about understanding intelligence. Also, it defines how automated reasoning procedures can make this knowledge available as needed.
- The **Planning and Execution** components depend on the analysis of knowledge representation and reasoning. Here, planning includes giving an initial state, finding their preconditions and effects, and a sequence of actions to achieve a state in which a particular goal holds.

Types of knowledge



Types of knowledge (Cont..)

- **Declarative Knowledge** – It includes concepts, facts, and objects expressed in a declarative sentence. It provides all the necessary information about the problem in terms of simple statements, either true or false.
- **Structural Knowledge** – It is a basic problem-solving knowledge that describes the relationship between concepts and objects.
- **Procedural Knowledge** – This is responsible for knowing how to do something and includes rules, strategies, procedures, etc.

  For example: Computer program.
- **Meta Knowledge** – Meta-knowledge is helpful in enhancing the efficiency of problem solving through proper reasoning process.
- **Heuristic Knowledge** – In this type, the knowledge representation is based on the strategies to solve the problems through the experience of past problems, compiled by an expert

Issues in Knowledge Representation

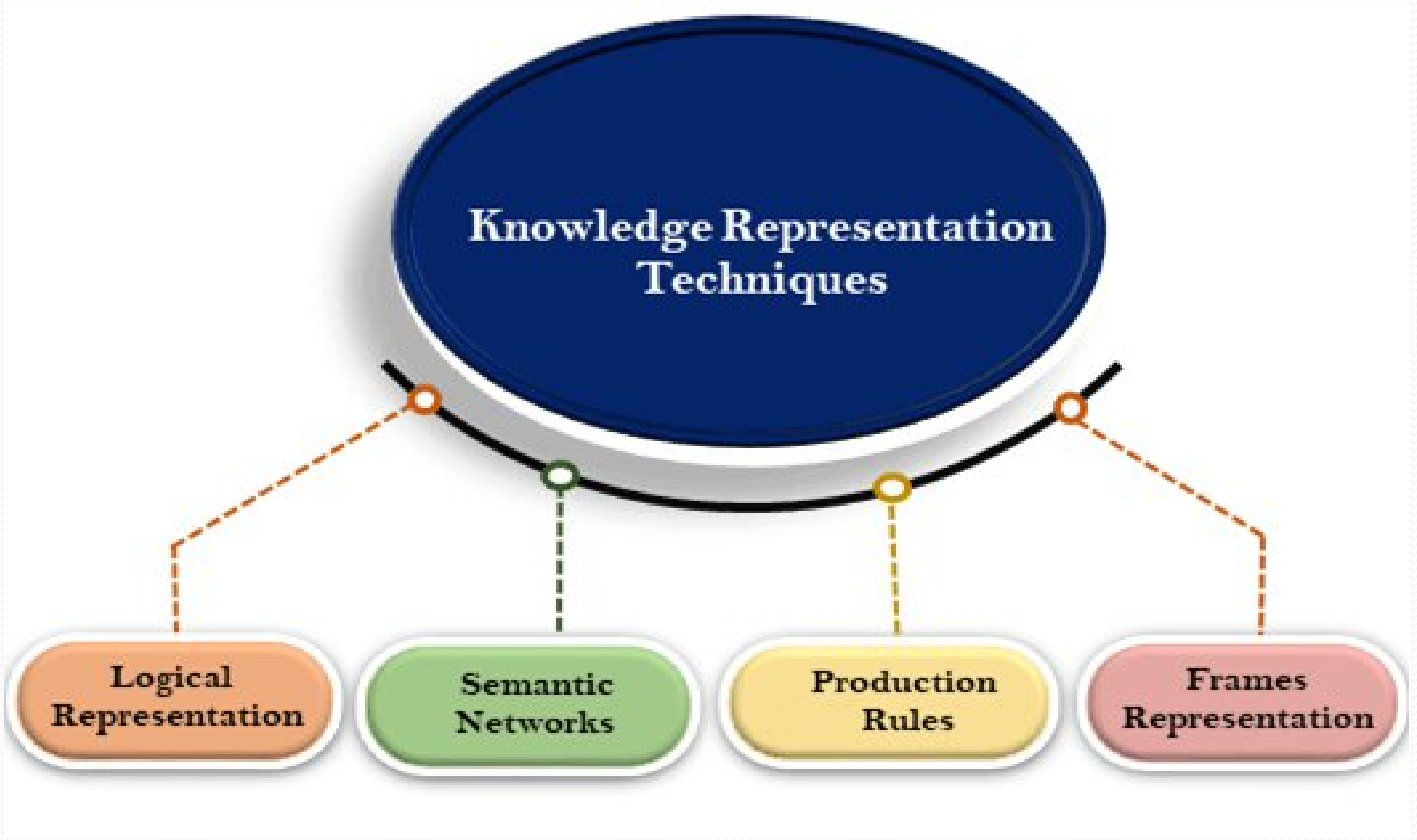
- **Ambiguity**: Information can be interpreted in different ways, leading to ambiguity in the knowledge representation.
  - Example: the word "bank" could refer to a financial institution, a riverbank, or even the act of tilting to one side. To avoid ambiguity, we need to consider the surrounding words and the overall context of the.

- **Incomplete information**:In AI and knowledge representation, incomplete information can arise when data is missing or when the system is unable to access all the relevant information
  - Example: Consider an AI system that helps doctors diagnose patients. If the system doesn't have access to a patient's full medical history, it may not be able to make an accurate diagnosis

- **Inconsistency**:Inconsistencies in information can arise when data from different sources conflict with each other.
  - Example : Let's say there is an AI system that helps to match job candidates with open positions. If the system receives conflicting information about a candidate's skills or work history from different sources, it may have trouble determining which information is correct. For example, if one reference says a candidate has excellent communication skills, but another reference says the candidate struggles to work in teams, the system may have difficulty making an accurate assessment.

- **Complexity**:Some domains may be so complex that representing all the relevant knowledge is difficult. This can be particularly challenging for AI systems that rely on structured data and require clear, well-defined relationships between different pieces of information.

Techniques of knowledge representation



1. Logical Representation

- Logical representation means drawing a conclusion based on various conditions
- Logical representation can be categorized into mainly two logics:
  - **i. Propositional Logic**
  - **ii. Predicate logic**

i. Propositional Logic:

- 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.

Ex: a) It is Sunday. b) The Sun rises from West (False proposition)

c) 3+3= 7(False proposition) d) 5 is a prime number.

There are two types of Propositions:

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

Ex: 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.

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

Ex: **a) "It is raining today, and street is wet."**

b) "Ankit is a doctor, and his clinic is in Mumbai."

- In compound proposition, there are set of symbols we use

  ¬ Negation (Ex: Today is Not Friday)

  V Disjunction (Ex: You should eat Or Watch TV at a time) [PVQ]

  ∧ Conjunction (Ex: You should eat And Watch TV at a time) [P ∧ Q]

  -> if - then (Ex: If there is rain then the roads are wet) [P->Q]

  <-> if and only if (Ex: I will go to Mall iff I have to do shopping) [P<-> Q]

P	Q	**¬P/¬Q**	PVQ	P∧Q	P->Q	P<-> Q
T	T	F/F	T	T	T	T
T	F	F/T	T	F	F	F
F	T	T/F	T	F	T	F
F	F	T/T	F	F	T	T

EXAMPLE: You can access the internet from campus only if you are cse students or you are not freshman

P -> (Q V ¬ R)

Limitations of Propositional logic:

- We cannot represent relations like ALL, some, or none with propositional logic.

Example:

- **All the girls are intelligent.**
- **Some apples are sweet.**

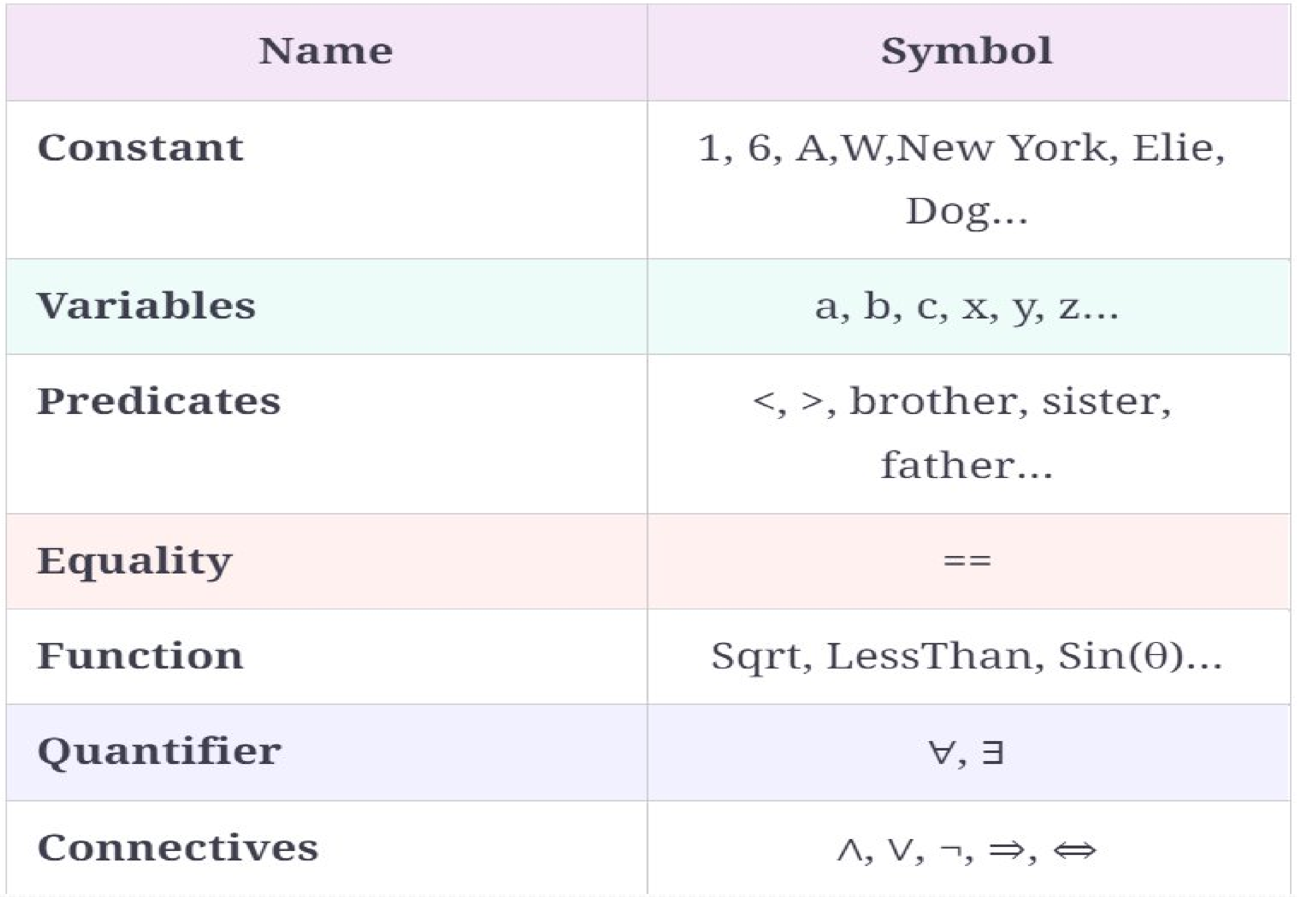
Predicate Logic

- Predicate logic can also be called as First Order Logic. It is an extension of Propositional Logic.
- FOL represents natural language statements in a concise way.
- It is a powerful language used to develop information about an object and express the relationship between objects.
- FOL not only assumes that does the world contains facts (like PL does), but it also assumes the following:
  1. **Objects**: A, B, people, numbers, colors, wars, theories, squares, pit, etc.
  2. **Relations**: It is unary relation such as red, round, sister of, brother of, etc.
  3. **Function**: father of, best friend, third inning of, end of, etc.

Predicate Logic

- First-order logic also has two main parts:

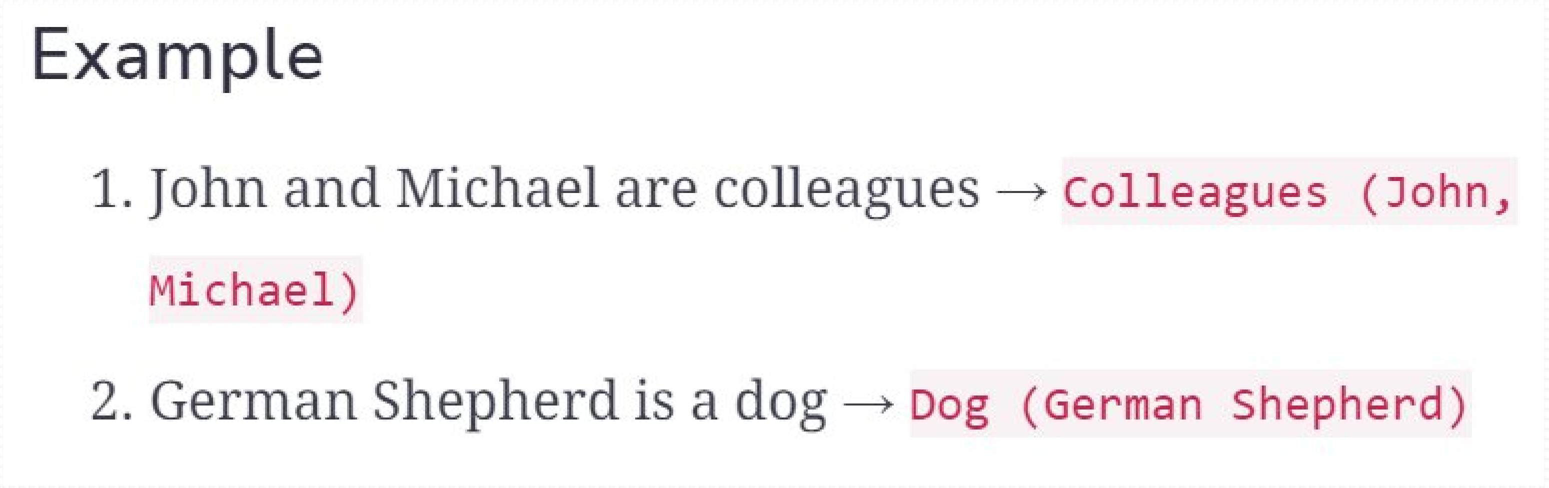
  i) Syntax ii) **Semantics**



Name	Symbol
Constant	1, 6, A,W,New York, Elie, Dog...
Variables	a, b, c, x, y, z...
Predicates	<, >, brother, sister, father...
Equality	==
Function	Sqrt, LessThan, Sin(θ)...
Quantifier	∀, ∃
Connectives	∧, ∨, ¬, ⇒, ⇔

Atomic and complex sentences in FOL

- **Atomic Sentence**
- This is a basic sentence of FOL formed from a predicate symbol followed by a parenthesis with a sequence of terms.
- We can represent atomic sentences as a predicate (value1, value2…., value n).



Example

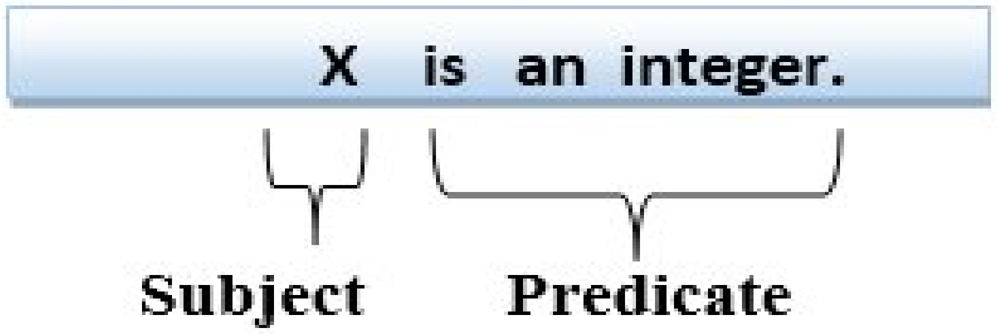
1. John and Michael are colleagues → `Colleagues (John, Michael)`
2. German Shepherd is a dog → `Dog (German Shepherd)`

Complex sentence: These are made by combining atomic sentences using connectives. It is mainly divided into two part:

- **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.

Example:

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 that allows us to specify how many objects or entities in a universe meet a certain condition.
- Quantifiers allow us to determine or identify the range and scope of the variable in a logical expression.
- There are two types of quantifier:
  - **Universal Quantifier (∀), (for all, for every, everyone, everything)**
  - **Existential quantifier(∃), (for some, at least one).**

Universal Quantifier:

Ex: All man drink coffee.

$\forall x$ man(x) → drink (x, coffee).

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

Existential quantifier:

Ex: Some boys are intelligent.

$\exists x$: boys(x) ∧ intelligent(x)

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

Note:

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

Some Examples of FOL using quantifier:

i) 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.

  $\forall x\ \mathbf{bird(x)} \rightarrow \mathbf{fly(x)}$.

ii) 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:

  $\forall x\ \mathbf{man(x)} \rightarrow \mathbf{respects\ (x, parent)}$.

iii) 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).

iv) 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)].**

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: $\forall x \exists (y)[P (x, y, z)]$, 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: $\forall x [A (x) B(y)]$, here x and y are the bound variables.

Some examples are:

1. Marcus was a man

Ans: man(Marcus)

2. Marcus was a Pompeian.

Ans: Pompeian(Marcus)

3. All Pompeian were Romans.

Ans: ∀x: Pompeian(x)-> Roman(x)

4. Caesar was a ruler.

Ans: ruler(Caesar)

5. All Romans were either loyal to Caesar or hated him

Ans: ∀x: Roman(x) -> loyalto (x, Caesar)V hate(x, Caesar)

6. Everyone is loyal to someone:

Ans: ∀x: ∃ y: loyalto (x, y)

7. People only try to assassinate rulers they are not loyal to.

Ans: ∀x: ∀y: person(x) ^ ruler(y) ^ tryassassinate(x,y) -> ¬ loyalto(x, y)

8. Marcus tried to assassinate Caesar.

Ans: tryassassinate (Marcus, Caesar)

9. All men are people

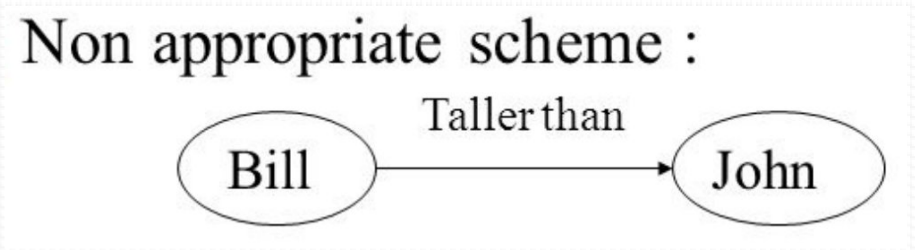
Ans: ∀x: man(x)-> person(x)

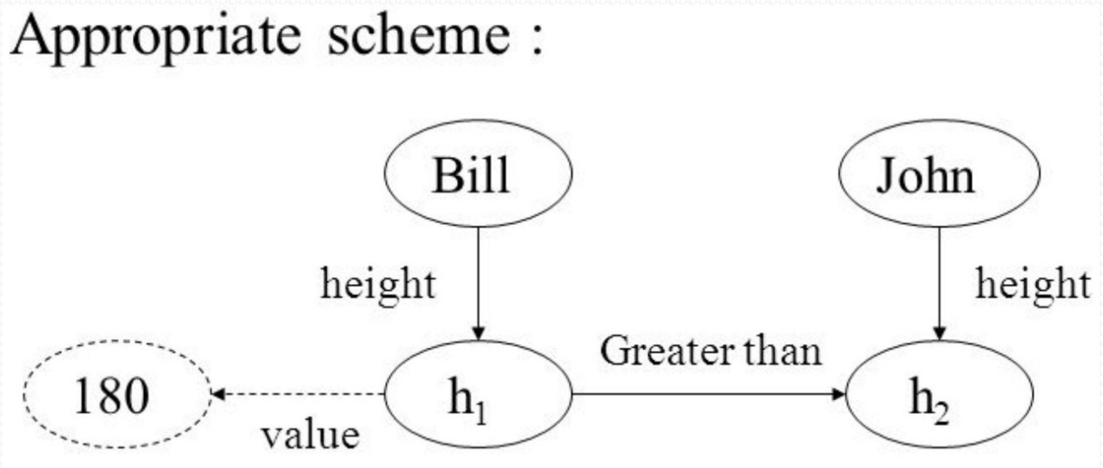
2. Semantic Network Representation

- A semantic Network is a simple notation scheme for logical knowledge representation.
- A SN consists of a concepts and relations between concepts.
- Representing a SN with directed graph
  - Vertices: denote concepts
  - Edges: represent relation between concepts.

Example

- Let us say a simple statement “Bill is taller than John”.

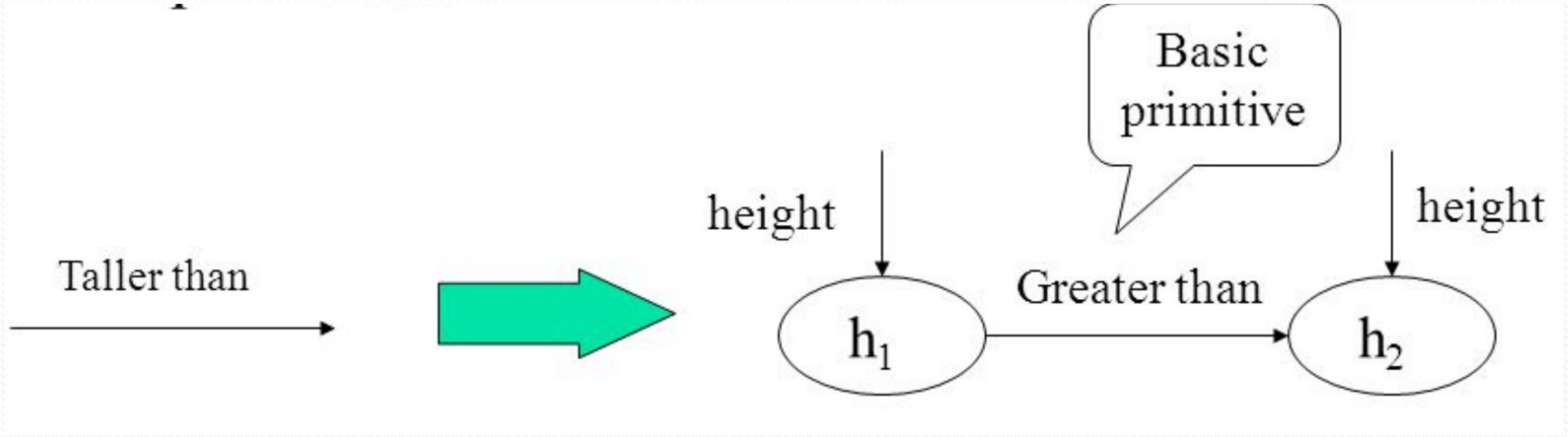




Semantic Networks Relations

For an appropriate scheme:

- Draw relations on the basis of primitives
- Represent complicated relations with this primitives.

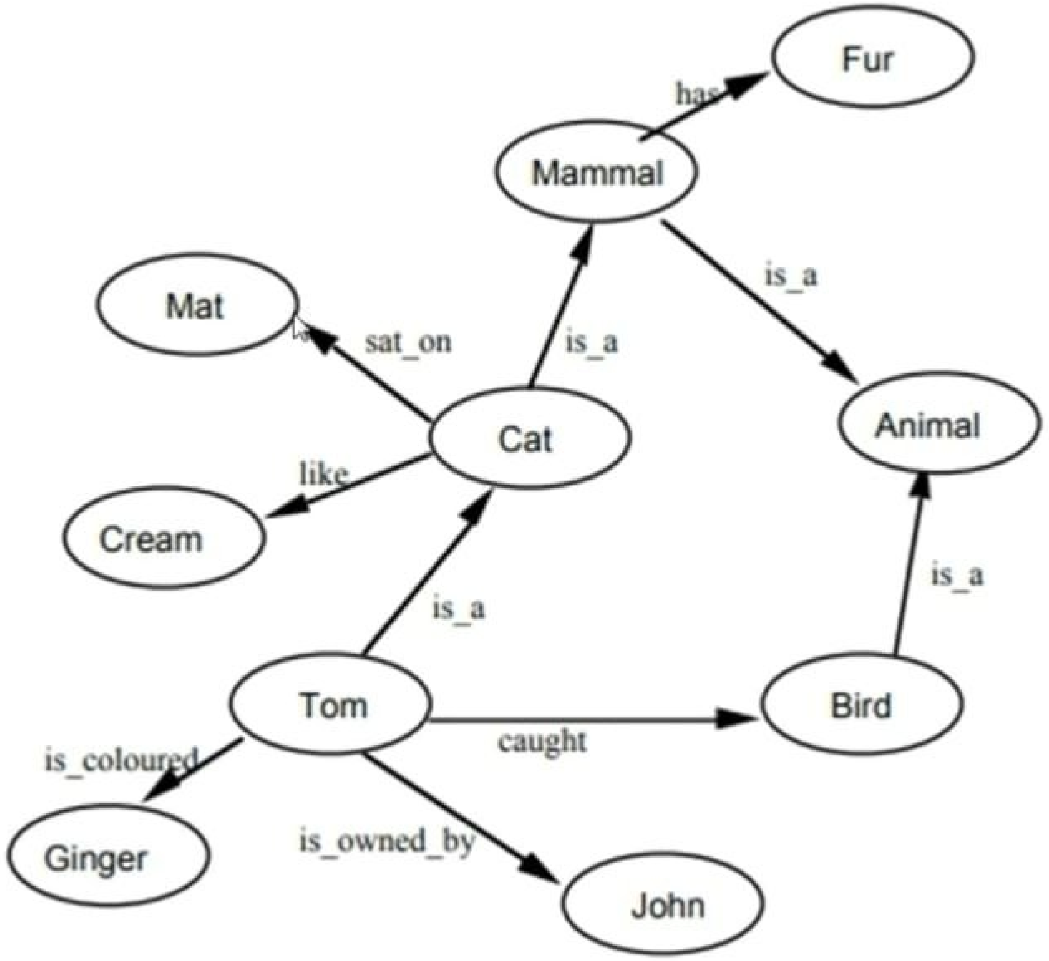


Example

Semantic Networks:

Example :

Tom is a cat.
Tom caught a bird.
Tom is owned by John.
Tom is ginger in colour.
Cats like cream.
The cat sat on the mat.
A cat is a mammal.
A bird is an animal.
All mammals are animals.
Mammals have fur.



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

Statements:

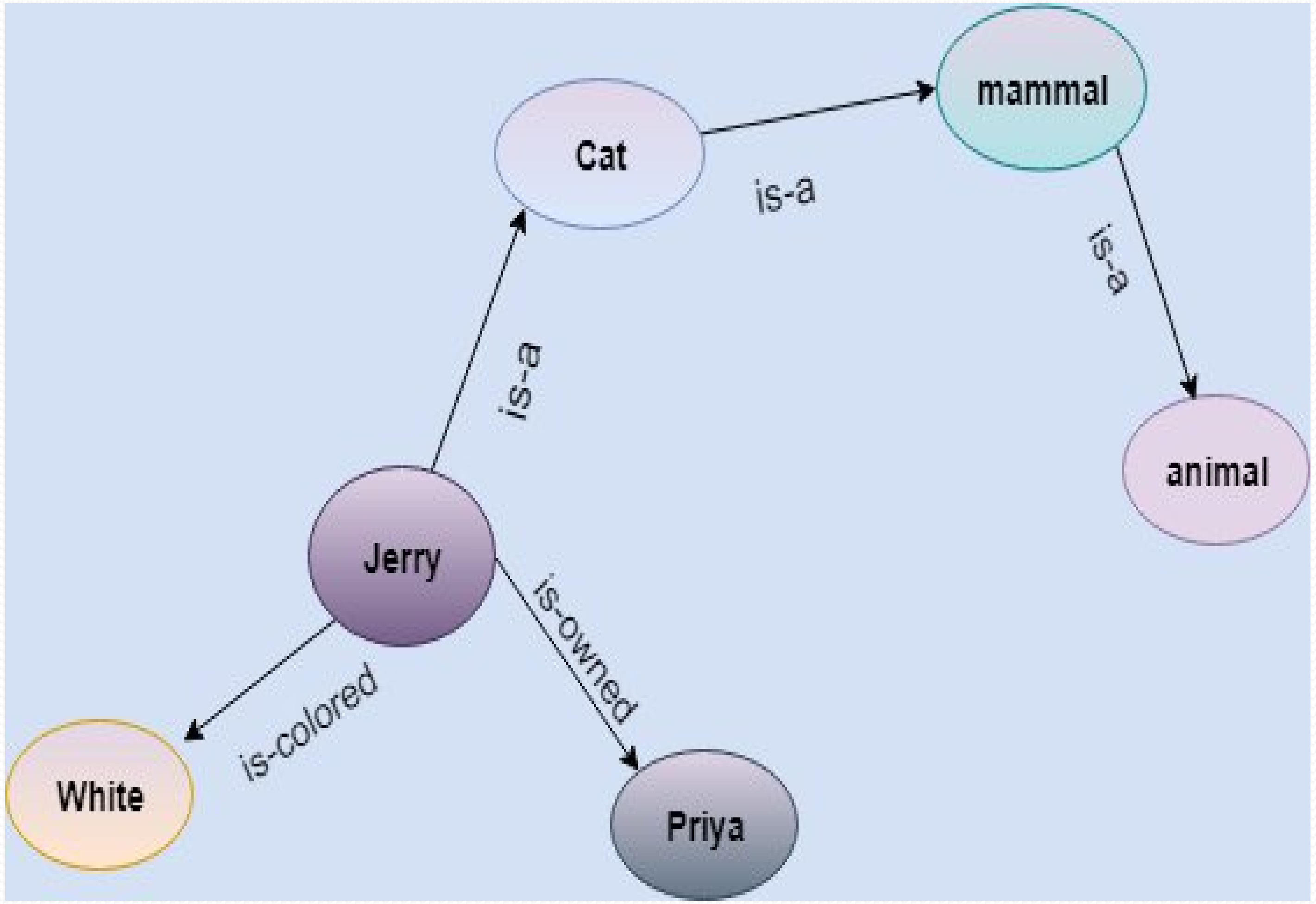
1.Jerry is a cat.

2.Jerry is a mammal

3.Jerry is owned by Priya.

4.Jerry is brown colored.

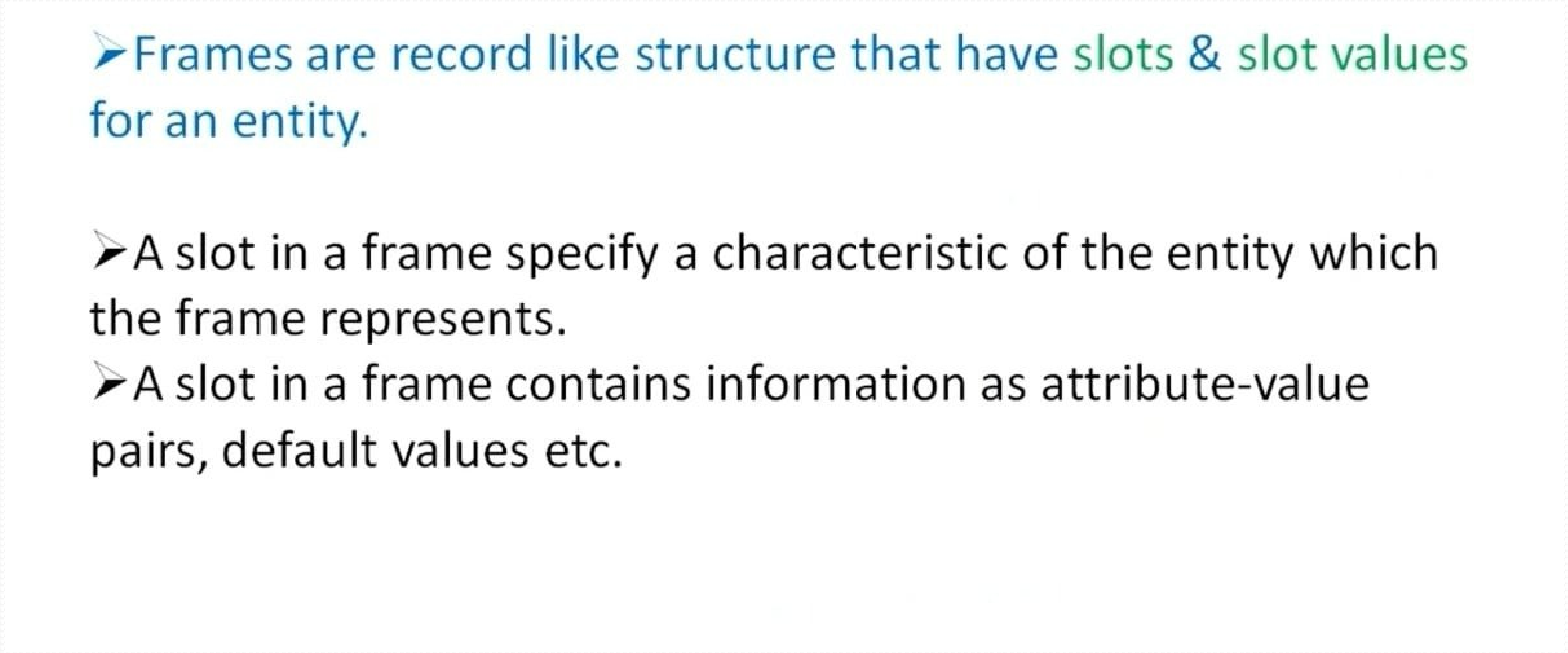
5.All Mammals are animal.



Drawbacks in Semantic representation:

1. 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.
2. Semantic networks try to model human-like memory to store the information, but in practice, it is not possible to build such a vast semantic network.
3. Semantic networks do not have any standard definition for the link names.
4. These networks are not intelligent and depend on the creator of the system.

3. Frame Representation



- Frames are record like structure that have slots & slot values for an entity.

- A slot in a frame specify a characteristic of the entity which the frame represents.
- A slot in a frame contains information as attribute-value pairs, default values etc.

Frames:

Example 1:

Employee Details

```
Raj Sharma(
        (Profession          (Value    Manager))
        (EmpID               (Value    100213))
        (Address             (Value    Delhi))
)
```

Frames:

Example 2:

"Tweety is a yellow bird having wings to fly"

```
Tweety(
        (Species                 (Value    bird))
        (color                   (Value    yellow))
        (Activity                (Value    Fly))
)
```

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
- 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).**

Predicate logic-Unification

- Unification is a process of matching and binding the variables in the predicates or expressions to create a single solution.

Example: P(x,F(y)) -------- 1

P(a, F(g(z))-------- 2

Solution: P[x/a , F(y)/F(g(z))] ----[y=g(z)]

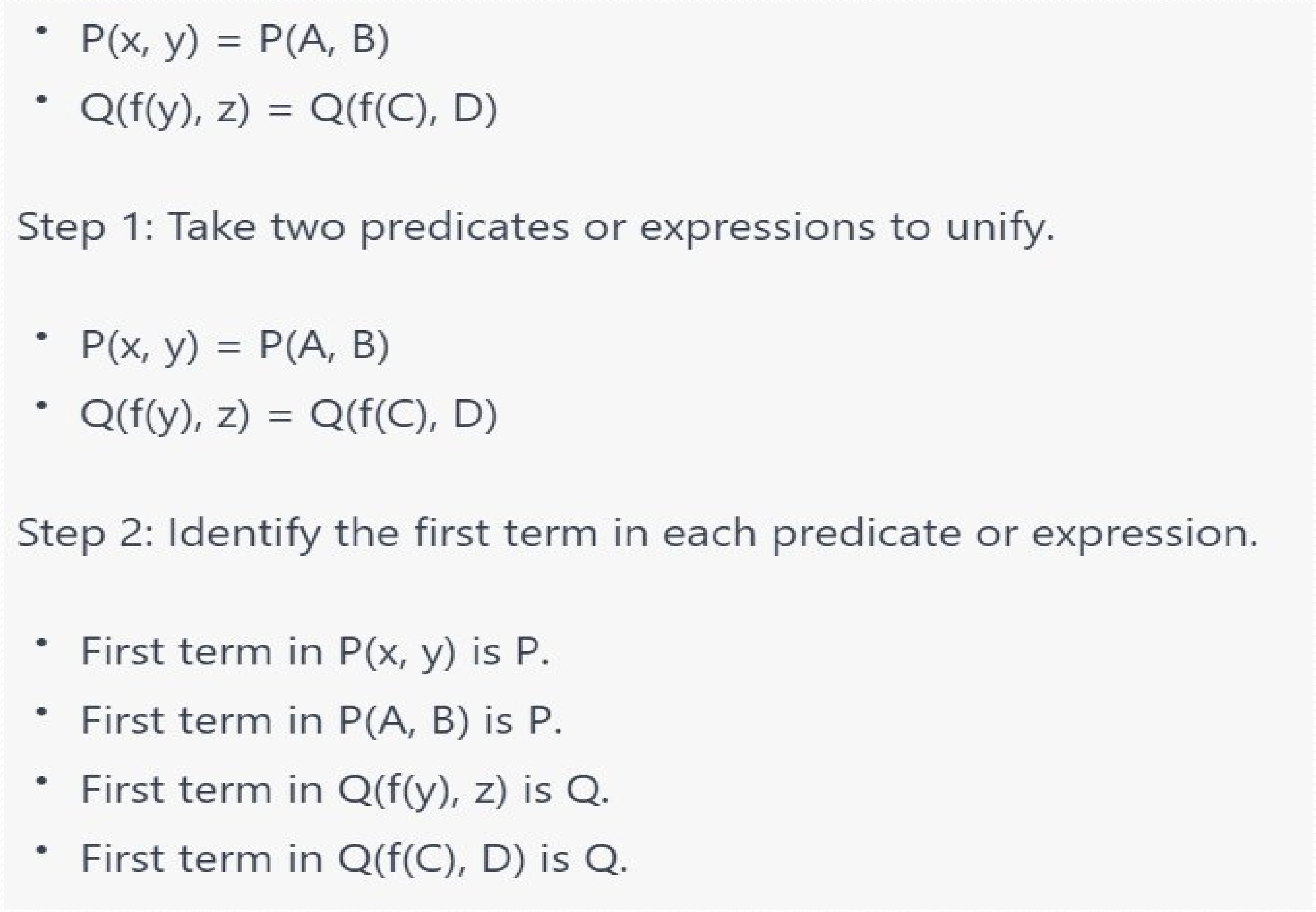
Conditions for unification:

- Atoms or expressions with various predicate symbols can never be united.
- Both phrases must have the same number of arguments.
- If two similar variables appear in the same expression, unification will fail.

Algorithm

1. Take two predicates or expressions to unify.
2. Identify the first term in each predicate or expression.
3. If both terms are the same, proceed to the next term. If they are different, continue to step 4.
4. Check if either term is a variable. If one term is a variable, bind the variable to the other term. If both terms are variables, choose one of the variables and bind it to the other variable.
5. Substitute the variable binding into both predicates or expressions.
6. Repeat steps 2 to 5 for each term in the predicates or expressions until all terms have been compared and unified.
7. If a contradiction is found during unification, terminate the unification process and return failure.
8. If unification is successful, return the unified predicates or expressions

Example:



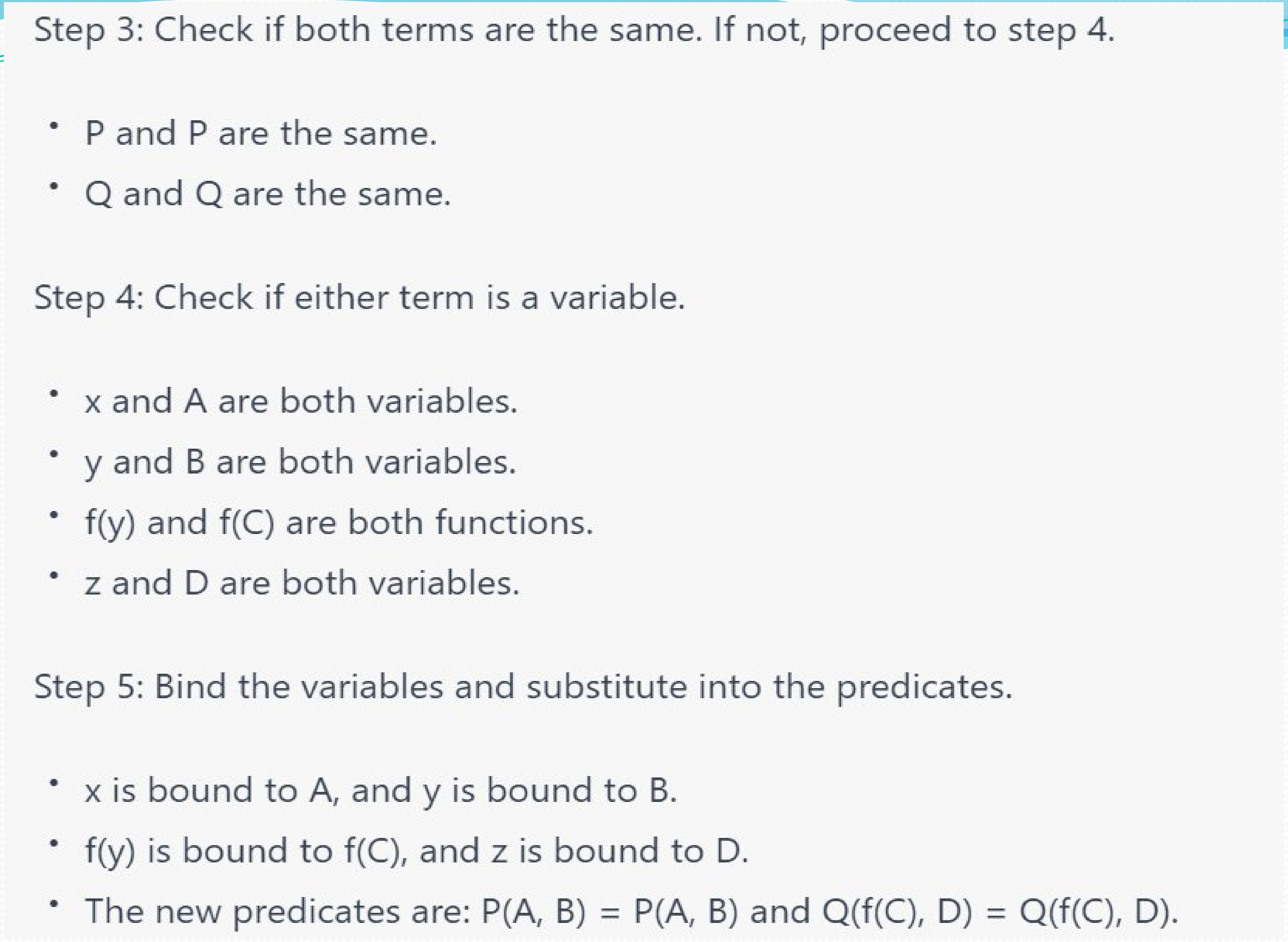
- P(x, y) = P(A, B)
- Q(f(y), z) = Q(f(C), D)

Step 1: Take two predicates or expressions to unify.

- P(x, y) = P(A, B)
- Q(f(y), z) = Q(f(C), D)

Step 2: Identify the first term in each predicate or expression.

- First term in P(x, y) is P.
- First term in P(A, B) is P.
- First term in Q(f(y), z) is Q.
- First term in Q(f(C), D) is Q.


Step 3: Check if both terms are the same. If not, proceed to step 4.

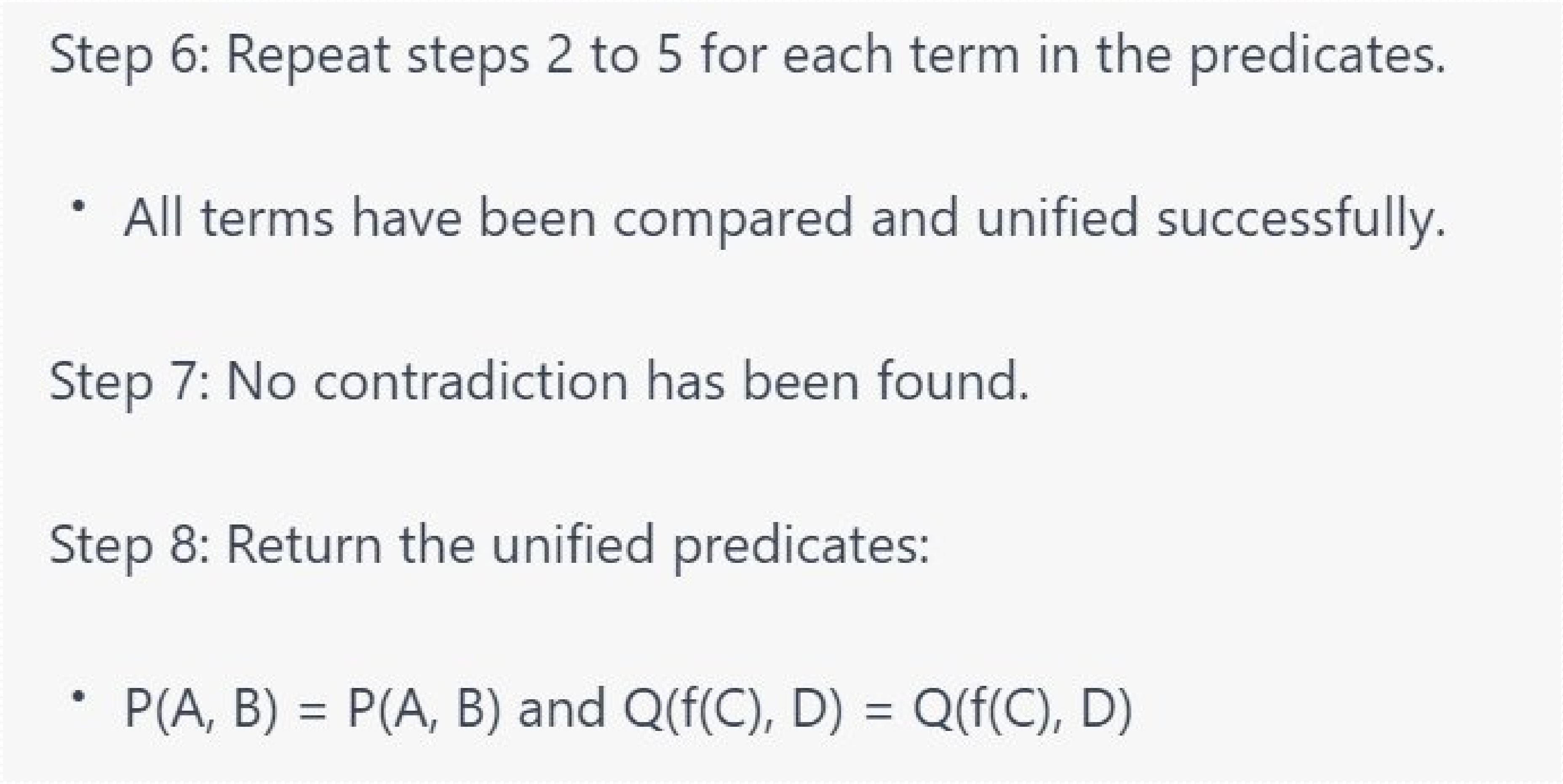
- P and P are the same.
- Q and Q are the same.

Step 4: Check if either term is a variable.

- x and A are both variables.
- y and B are both variables.
- f(y) and f(C) are both functions.
- z and D are both variables.

Step 5: Bind the variables and substitute into the predicates.

- x is bound to A, and y is bound to B.
- f(y) is bound to f(C), and z is bound to D.
- The new predicates are: P(A, B) = P(A, B) and Q(f(C), D) = Q(f(C), D).



Step 6: Repeat steps 2 to 5 for each term in the predicates.

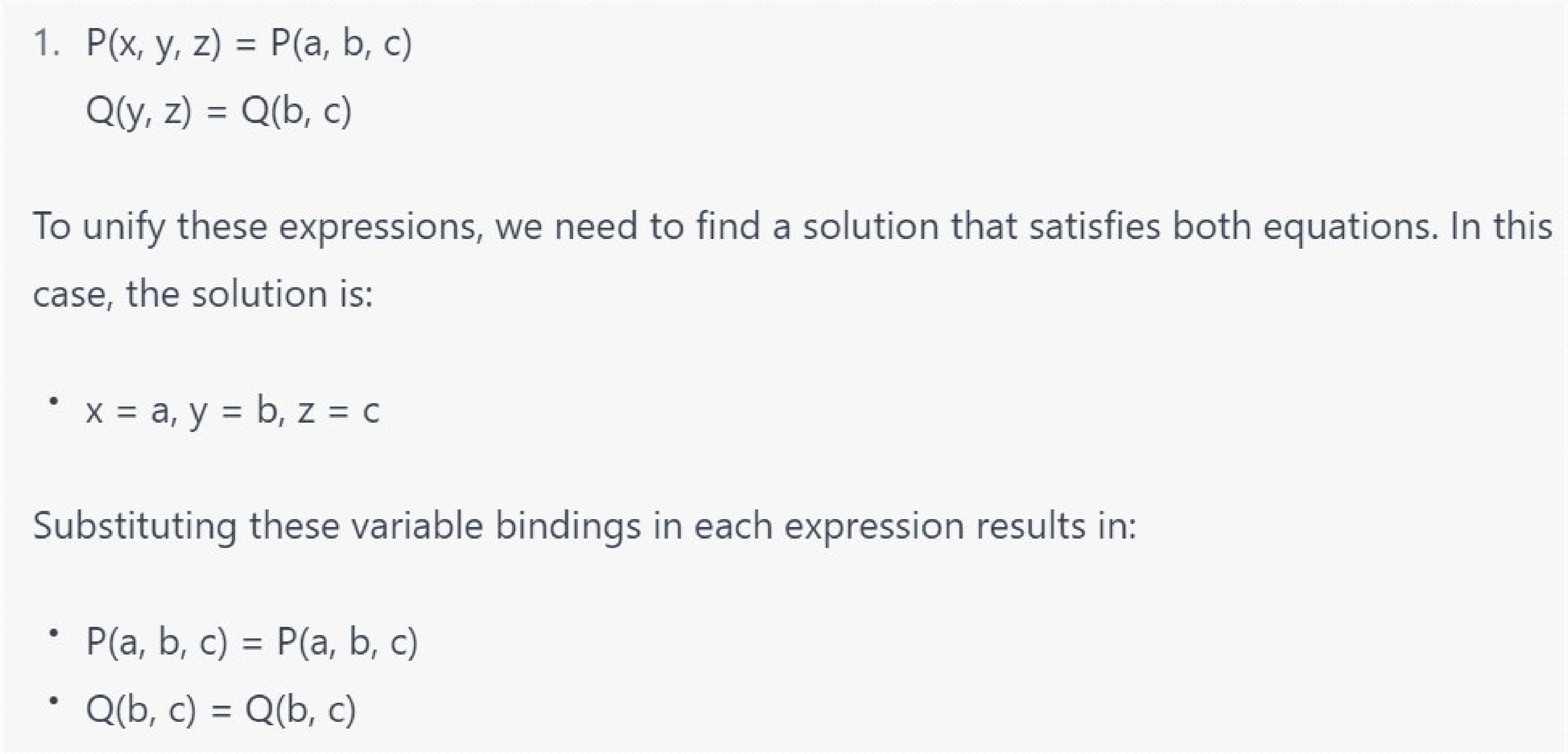
- All terms have been compared and unified successfully.

Step 7: No contradiction has been found.

Step 8: Return the unified predicates:

- P(A, B) = P(A, B) and Q(f(C), D) = Q(f(C), D)

Examples



1. $P(x, y, z) = P(a, b, c)$

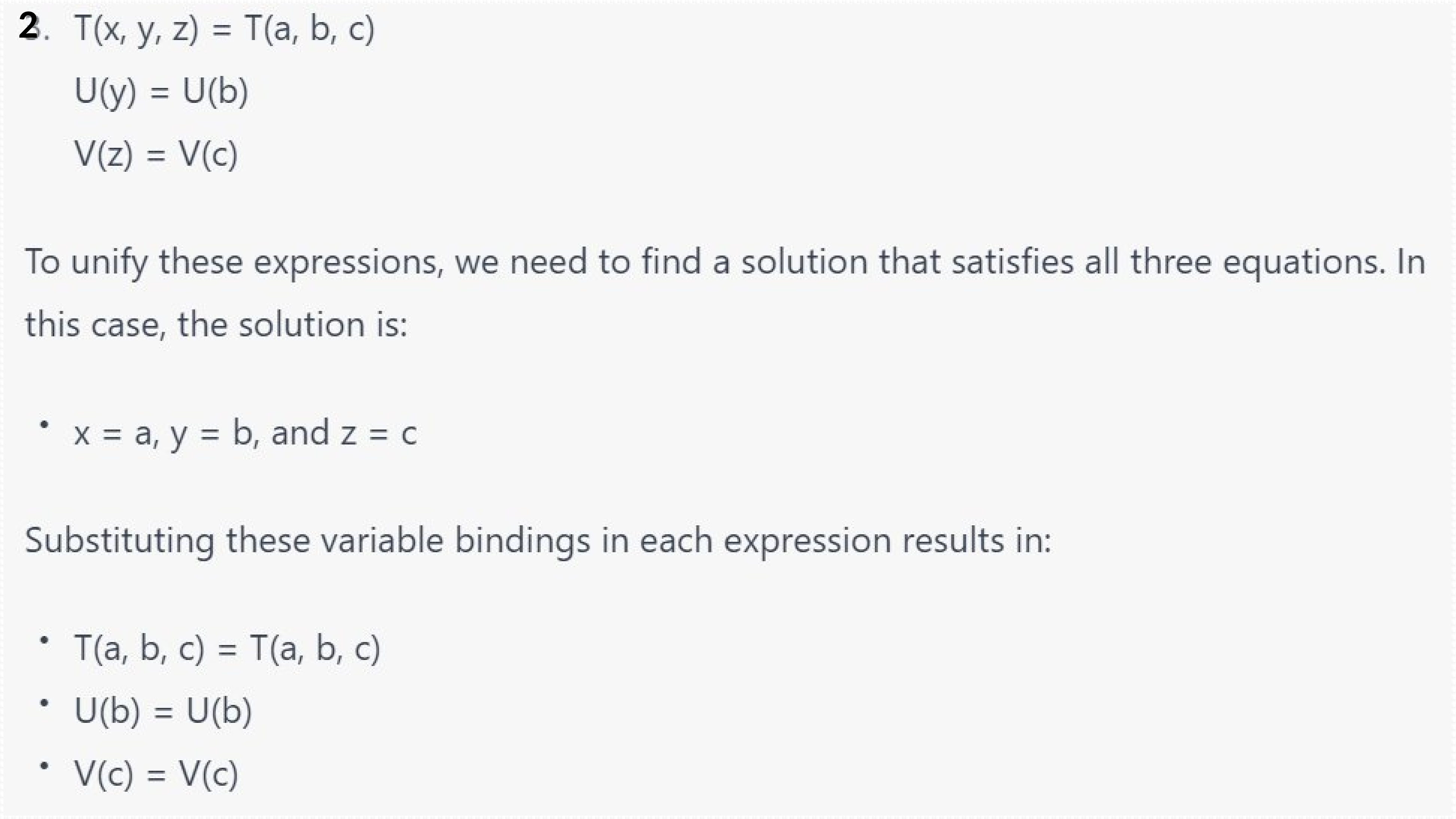
   $Q(y, z) = Q(b, c)$

To unify these expressions, we need to find a solution that satisfies both equations. In this case, the solution is:

- $x = a, y = b, z = c$

Substituting these variable bindings in each expression results in:

- $P(a, b, c) = P(a, b, c)$
- $Q(b, c) = Q(b, c)$



2. T(x, y, z) = T(a, b, c)

   U(y) = U(b)

   V(z) = V(c)

To unify these expressions, we need to find a solution that satisfies all three equations. In this case, the solution is:

- x = a, y = b, and z = c

Substituting these variable bindings in each expression results in:

- T(a, b, c) = T(a, b, c)
- U(b) = U(b)
- V(c) = V(c)

Predicate Logic - Resolution

- Resolution method is an inference rule which is used in both Propositional as well as First-order Predicate Logic in different ways.
- In resolution method, we use **Proof by contradiction** technique to prove the given statement.
- The key idea for the resolution method is to use the knowledge base and negated goal to obtain null clause (which indicates contradiction).
- Since the knowledge base itself is consistent, the contradiction must be introduced by a negated goal.
- As a result, we have to conclude that the original goal is true.

Steps for Resolution

1. Conversion of facts into first-order logic.
2. Convert FOL statements into CNF (Conjunctive Normal Form)
   - In First order logic resolution, it is required to convert the FOL into CNF as CNF form makes easier for resolution proofs.
   - Eliminate all implication (→) and rewrite
   - Move negation (¬)inwards and rewrite
   - Rename variables or standardize variables
   - Eliminate existential instantiation quantifier by elimination
   - Drop Universal quantifiers
   - Distribute conjunction ∧ over disjunction ¬.
3. Negate the statement which needs to prove (proof by contradiction)
4. Draw resolution graph (unification).

Conjunctive Normal Form

- In propositional logic, the resolution method is applied only to those clauses which are disjunction of literals.

There are following steps used to convert into CNF:

1) Eliminate bi-conditional implication by replacing A ? B with (A ? B) ? (B ?A)

2) Eliminate implication by replacing A -> B with ¬A V B.

3) In CNF, negation(¬) appears only in literals, therefore we move it inwards as:

- **¬ (¬A) = A** (double-negation elimination
- **¬ (A ^ B) = (¬A V ¬B)** (De Morgan)
- **¬(A V B) = (¬A ^ ¬B)** (De Morgan)

4) Finally, using distributive law on the sentences, and form the CNF as:

(A1 V B1) ^ (A2 V B2) ^ …. ^ (An V Bn).

Note: CNF can also be described as AND of ORS

Example:

Let us say:

{Bird(F(x)) V Loves(G(x), x)} and {¬Loves(a, b) V ¬Kills(a, b)}

Eliminate the complementary literals

"Loves(G(x),x) and Loves(a,b))" with ={a/G(x), b/x} to give the following output clause:

{Bird(F(x)) V ¬Kills(G(x),x)}

The rule applied on the following example is called **Binary Resolution** as it has solved exactly two literals. But, binary resolution is not complete. An alternative approach is to extend the **factoring** i.e., to remove redundant literals to the first order case. Thus, the combination of binary resolution and factoring is complete.

Convert each statement into CNF

1. (P->Q) ->Q

= (~P V Q) -> Q

= ~(~P V Q) V Q

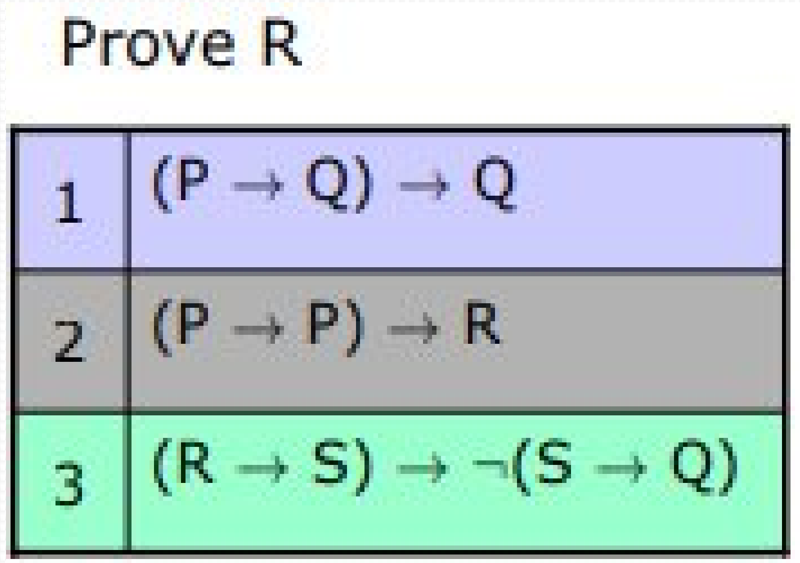
= (~~P ^ ~Q) V Q

= (P ^ ~Q) V Q

= (P V Q) ^ (~Q V Q)

(P V Q)

[(P->Q) =



Prove R

1	$(P \to Q) \to Q$
2	$(P \to P) \to R$
3	$(R \to S) \to \neg(S \to Q)$

2. (P -> P) -> R

= ~(~P V P) V R

= (P ^ ~P) V R

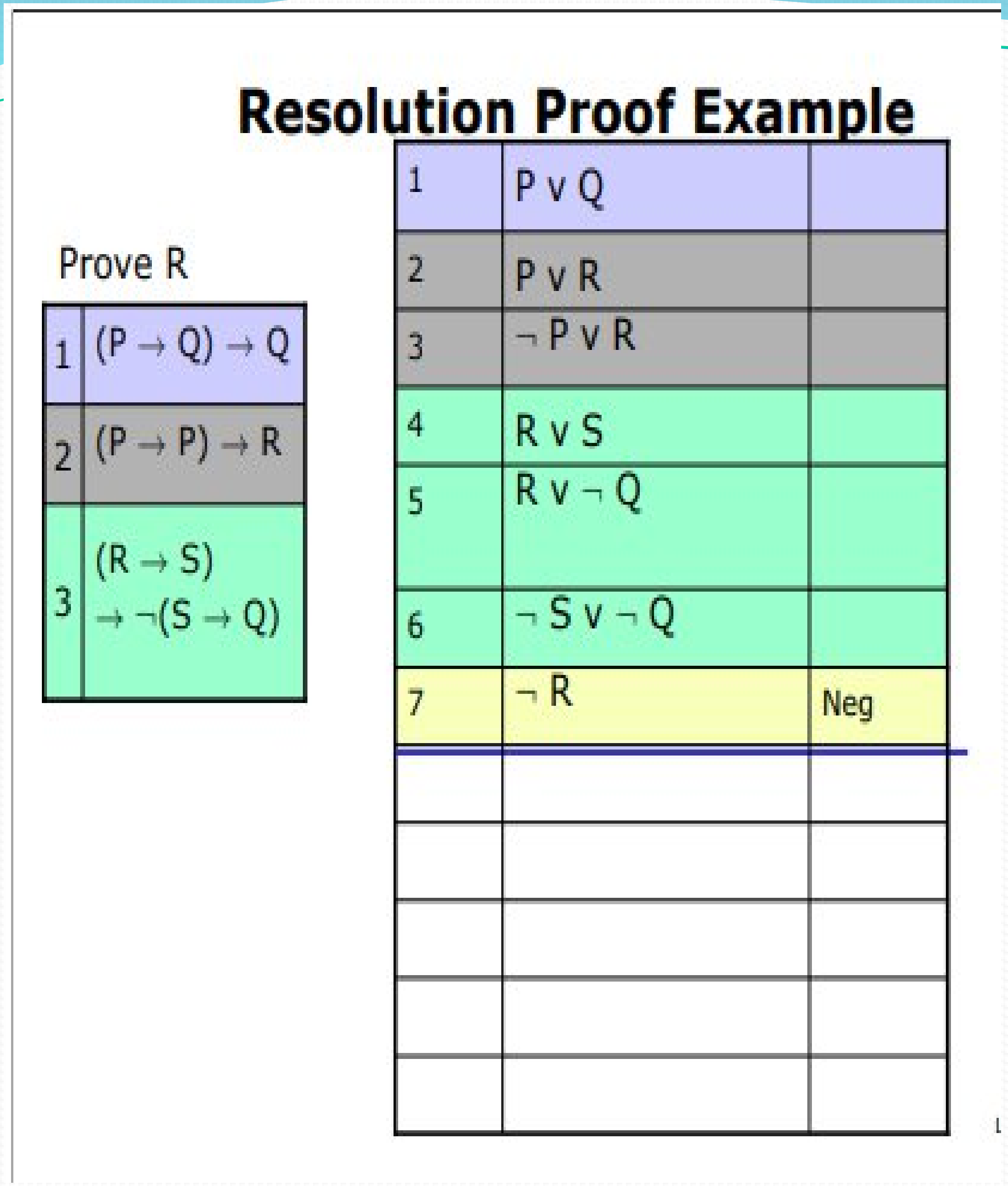
= (P V R) ^ (~P V R)

3. (R -> S) -> ~(S -> Q)

= (~R V S) - > ~(~S V Q)

= ~(~R V S) V ~(~S V Q)

= (R V S) ^ (R V ~Q) ^ (~S V ~Q)

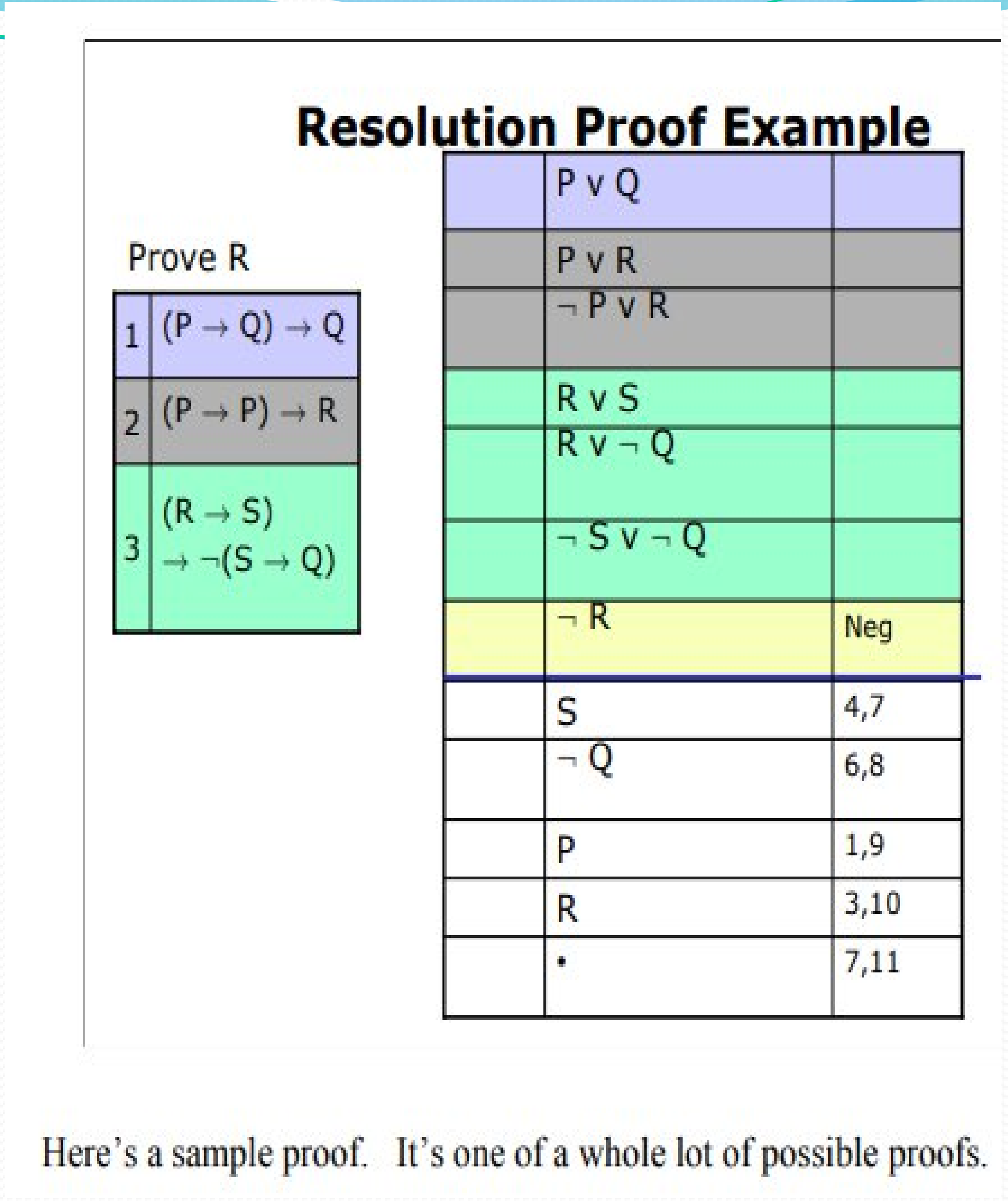


Resolution Proof Example

Prove R

1	(P → Q) → Q
2	(P → P) → R
3	(R → S) → ¬(S → Q)

1	P v Q	
2	P v R	
3	¬ P v R	
4	R v S	
5	R v ¬ Q	
6	¬ S v ¬ Q	
7	¬ R	Neg



Resolution Proof Example

Prove R

1	(P → Q) → Q
2	(P → P) → R
3	(R → S) → ¬(S → Q)

	P v Q	
	P v R	
	¬ P v R	
	R v S	
	R v ¬ Q	
	¬ S v ¬ Q	
	¬ R	Neg
	S	4,7
	¬ Q	6,8
	P	1,9
	R	3,10
	•	7,11

Here's a sample proof. It's one of a whole lot of possible proofs.

Resolution Proof Example.

(a) Marcus was a man.

(b) Marcus was a Roman.

(c) All men are people.

(d) Caesar was a ruler.

(e) All Romans were either loyal to Caesar or hated him (or both).

(f) Everyone is loyal to someone.

(g) People only try to assassinate rulers they are not loyal to.

(h) Marcus tried to assassinate Caesar.

Convert to First order Logic.

(a) Marcus was a man.

(b) Marcus was a Roman.

(c) All men are people.

(d) Caesar was a ruler.

(e) All Romans were either loyal to Caesar or hated him (or both).

(f) Everyone is loyal to someone.

(g) People only try to assassinate rulers they are not loyal to.

(h) Marcus tried to assassinate Caesar.

(a) man(marcus)

Convert to First order Logic

(a) Marcus was a man.

(b) Marcus was a Roman.

(c) All men are people.

(d) Caesar was a ruler.

(e) All Romans were either loyal to Caesar or hated him (or both).

(f) Everyone is loyal to someone.

(g) People only try to assassinate rulers they are not loyal to.

(h) Marcus tried to assassinate Caesar.

(a) man(marcus)

(b) roman(marcus)

(c) $\forall$X. man(X) $\rightarrow$ person(X)

(d) ruler(caesar)

(e) $\forall$X. roman(x) $\rightarrow$ loyal(X,caesar) $\vee$ hate(X,caesar)

(f) $\forall$X$\exists$Y. loyal(X,Y)

(g) $\forall$X$\forall$Y. person(X) $\wedge$ ruler(Y) tryassasin(X,Y) $\rightarrow$ $\neg$loyal(X,Y)

(h) tryassasin(marcus,caesar)

Convert to Clausal Form

(a) man(marcus)

(b) roman(marcus)

(c) $\forall X.\ man(X) \rightarrow person(X)$

(d) ruler(caesar)

(e) $\forall X.\ roman(x) \rightarrow loyal(X,caesar) \vee hate(X,caesar)$

(f) $\forall X \exists Y.\ loyal(X,Y)$

(g) $\forall X \forall Y.\ person(X) \wedge ruler(Y) \wedge tryassasin(X,Y) \rightarrow \neg loyal(X,Y)$

(h) tryassasin(marcus,caesar)

Convert to Clausal Form

(a) man(marcus)

(b) roman(marcus)

(c) $\forall X.\ man(X) \rightarrow person(X)$

$(\neg man(X), person(X))$

(d) ruler(caesar)

(e) $\forall X.\ roman(x) \rightarrow loyal(X,caesar) \vee hate(X,caesar)$

(f) $\forall X \exists Y.\ loyal(X,Y)$

(g) $\forall X \forall Y.\ person(X) \wedge ruler(Y) \wedge tryassasin(X,Y) \rightarrow \neg loyal(X,Y)$

(h) tryassasin(marcus,caesar)

Convert to Clausal Form

(a) man(marcus)
(b) roman(marcus)
(c) $\forall$X. man(X) $\rightarrow$ person(X)
(¬man(X), person(X))
(d) ruler(caesar)
(e) $\forall$X. roman(x) $\rightarrow$ loyal(X,caesar) $\vee$ hate(X,caesar)
(f) $\forall$X$\exists$Y. loyal(X,Y)
(g) $\forall$X$\forall$Y. person(X) $\wedge$ ruler(Y) $\wedge$ tryassasin(X,Y) $\rightarrow$ ¬loyal(X,Y)
(h) tryassasin(marcus,caesar)

Convert to Clausal Form

(a) man(marcus)

(b) roman(marcus)

(c) $\forall X.\ man(X) \rightarrow person(X)$

$(\neg man(X), person(X))$

(d) ruler(caesar)

(e) $\forall X.\ roman(X) \rightarrow loyal(X,caesar) \vee hate(X,caesar)$

$(\neg roman(X), loyal(X,caesar), hate(X,caesar))$

(f) $\forall X \exists Y.\ loyal(X,Y)$

$(loyal(X,f(X))$

(g) $\forall X \forall Y.\ person(X) \wedge ruler(Y) \wedge tryassasin(X,Y) \rightarrow \neg loyal(X,Y)$

(h) tryassasin(marcus,caesar)

Convert to Clausal Form

(a) man(marcus)
(b) roman(marcus)
(c) $\forall X.\ man(X) \rightarrow person(X)$
    $(\neg man(X),\ person(X))$
(d) ruler(caesar)
(e) $\forall X.\ roman(X) \rightarrow loyal(X,caesar) \vee hate(X,caesar)$
    $(\neg roman(X),\ loyal(X,caesar),\ hate(X,caesar))$
(f) $\forall X \exists Y.\ loyal(X,Y)$
    $(loyal(X,f(X)))$
(g) $\forall X \forall Y.\ person(X) \wedge ruler(Y) \wedge tryassasin(X,Y) \rightarrow \neg loyal(X,Y)$
    $(\neg person(X),\ \neg ruler(Y),\ \neg tryassasin(X,Y),\ \neg loyal(X,Y))$
(h) tryassasin(marcus,caesar)

Convert to Clausal Form

(a) man(marcus)

(b) roman(marcus)

(c) $\forall X.$ man(X) $\rightarrow$ person(X)

(¬man(X), person(X))

(d) ruler(caesar)

(e) $\forall X.$ roman(X) $\rightarrow$ loyal(X,caesar) $\vee$ hate(X,caesar)

(¬roman(X), loyal(X,caesar), hate(X,caesar))

(f) $\forall X \exists Y.$ loyal(X,Y)

(loyal(X,f(X))

(g) $\forall X \forall Y.$ person(X) $\wedge$ ruler(Y) $\wedge$ tryassasin(X,Y) $\rightarrow$ ¬loyal(X,Y)

(¬person(X), ¬ruler(Y), ¬tryassasin(X,Y), ¬loyal(X,Y))

(h) tryassasin(marcus,caesar)

Convert to Clausal Form

1. man(marcus)
2. roman(marcus)
3. (¬man(X), person(X))
4. ruler(caesar)
5. (¬roman(X), loyal(X,caesar), hate(X,caesar))
6. (loyal(X,f(X))
7. (¬person(X), ¬ruler(Y), ¬tryassasin(X,Y), ¬loyal(X,Y))
8. tryassasin(marcus,caesar)

Query

Who hated Caesar?

In First Order logic.

1. $\exists X.$ hate(X,caesar)

Negate!

1. $\forall X. \neg$hate(X,caesar)

Query

Who hated Caesar?

In First Order logic.

1. $\exists X.$ hate(X,caesar)

Negate!

1. $\forall X. \neg$hate(X,caesar)

Clausal Form, with answer literal.
9. ($\neg$hate(X,caesar), ans(X))

Resolution Proof

1. man(marcus)
2. roman(marcus)
3. (¬man(X), person(X))
4. ruler(caesar)
5. (¬roman(X), loyal(X,caesar), hate(X,caesar))
6. (loyal(X,f(X))
7. (¬person(X), ¬ruler(Y), ¬tryassasin(X,Y), ¬loyal(X,Y))
8. tryassasin(marcus,caesar)
9. ¬hate(X,caesar)
10. R[9,5c] (¬roman(X), loyal(X,caesar), ans(X))

Resolution Proof

1. man(marcus)
2. roman(marcus)
3. (¬man(X), person(X))
4. ruler(caesar)
5. (¬roman(X), loyal(X,caesar), hate(X,caesar))
6. (loyal(X,f(X))
7. (¬person(X), ¬ruler(Y), ¬tryassasin(X,Y), ¬loyal(X,Y))
8. tryassasin(marcus,caesar)
9. ¬hate(X,caesar)
10. R[9,5c] (¬roman(X), loyal(X,caesar), ans(X))
11. R[10a,2] {X=marcus} (loyal(markus,caesar), ans(markus))

Resolution Proof

1. man(marcus)
2. roman(marcus)
3. (¬man(X), person(X))
4. ruler(caesar)
5. (¬roman(X), loyal(X,caesar), hate(X,caesar))
6. (loyal(X,f(X))
7. (¬person(X), ¬ruler(Y), ¬tryassasin(X,Y), ¬loyal(X,Y))
8. tryassasin(marcus,caesar)
9. ¬hate(X,caesar)
10. R[9,5c] (¬roman(X), loyal(X,caesar), ans(X))
11. R[10a,2] {X=marcus} (loyal(markus,caesar), ans(markus))
12. R[11,7c] {X=markus, Y=caesar} (¬person(markus), ¬ruler(caesar), ¬tryassasin(markus,caesar), ans(markus))

Resolution Proof

1. man(marcus)
2. roman(marcus)
3. (¬man(X), person(X))
4. ruler(caesar)
5. (¬roman(X), loyal(X,caesar), hate(X,caesar))
6. (loyal(X,f(X))
7. (¬person(X), ¬ruler(Y), ¬tryassasin(X,Y), ¬loyal(X,Y))
8. tryassasin(marcus,caesar)
9. ¬hate(X,caesar)
10. R[9,5c] (¬roman(X), loyal(X,caesar), ans(X))
11. R[10a,2] {X=marcus} (loyal(markus,caesar), ans(markus))
12. R[11,7c] {X=markus, Y=caesar} (¬person(markus), ¬ruler(caesar), ¬tryassasin(markus,caesar), ans(markus))
13. R[12a,3b] {X=markus} (¬man(markus), ¬ruler(caesar), ¬tryassasin(markus,caesar), ans(markus))

Resolution Proof

1. man(marcus)
2. roman(marcus)
3. (¬man(X), person(X))
4. ruler(caesar)
5. (¬roman(X), loyal(X,caesar), hate(X,caesar))
6. (loyal(X,f(X))
7. (¬person(X), ¬ruler(Y), ¬tryassasin(X,Y), ¬loyal(X,Y))
8. tryassasin(marcus,caesar)
9. ¬hate(X,caesar)
10. R[9,5c] (¬roman(X), loyal(X,caesar), ans(X))
11. R[10a,2] {X=marcus} (loyal(markus,caesar), ans(markus))
12. R[11,7c] {X=markus, Y=caesar} (¬person(markus), ¬ruler(caesar), ¬tryassasin(markus,caesar), ans(markus))
13. R[12a,3b] {X=markus} (¬man(markus), ¬ruler(caesar), ¬tryassasin(markus,caesar), ans(markus))
14. R[13a,1] (¬ruler(caesar), ¬tryassasin(markus,caesar), ans(markus))

Resolution Proof

1. man(marcus)
2. roman(marcus)
3. (¬man(X), person(X))
4. ruler(caesar)
5. (¬roman(X), loyal(X,caesar), hate(X,caesar))
6. (loyal(X,f(X))
7. (¬person(X), ¬ruler(Y), ¬tryassasin(X,Y), ¬loyal(X,Y))
8. tryassasin(marcus,caesar)
9. ¬hate(X,caesar)
10. R[9,5c] (¬roman(X), loyal(X,caesar), ans(X))
11. R[10a,2] {X=marcus} (loyal(markus,caesar), ans(markus))
12. R[11,7c] {X=markus, Y=caesar} (¬person(markus), ¬ruler(caesar), ¬tryassasin(markus,caesar), ans(markus))
13. R[12a,3b] {X=markus} (¬man(markus), ¬ruler(caesar), ¬tryassasin(markus,caesar), ans(markus))
14. R[13a,1] (¬ruler(caesar), ¬tryassasin(markus,caesar), ans(markus))
15. R[13a,2] (¬tryassasin(markus,caesar), ans(markus))

Resolution Proof

1. man(marcus)
2. roman(marcus)
3. (¬man(X), person(X))
4. ruler(caesar)
5. (¬roman(X), loyal(X,caesar), hate(X,caesar))
6. (loyal(X,f(X))
7. (¬person(X), ¬ruler(Y), ¬tryassasin(X,Y), ¬loyal(X,Y))
8. tryassasin(marcus,caesar)
9. ¬hate(X,caesar)
10. R[9,5c] (¬roman(X), loyal(X,caesar), ans(X))
11. R[10a,2] {X=marcus} (loyal(markus,caesar), ans(markus))
12. R[11,7c] {X=markus, Y=caesar} (¬person(markus), ¬ruler(caesar), ¬tryassasin(markus,caesar), ans(markus))
13. R[12a,3b] {X=markus} (¬man(markus), ¬ruler(caesar), ¬tryassasin(markus,caesar), ans(markus))
14. R[13a,1] (¬ruler(caesar), ¬tryassasin(markus,caesar), ans(markus))
15. R[13a,2] (¬tryassasin(markus,caesar), ans(markus))
16. R[15a,8] ans(markus)

Example of FOPL resolution

Consider the following knowledge base:

1. Gita likes all kinds of food.
2. Mango and chapati are food.
3. Gita eats almond and is still alive.
4. Anything eaten by anyone and is still alive is food.

Goal: Gita likes almond.

Solution: Convert the given sentences into FOPL as:

Let, x be the light sleeper.

1. ?x: food(x) ? likes(Gita,x)
2. food(Mango),food(chapati)
3. ?x?y: eats(x,y) ? ¬ killed(x ? food(y)
4. eats(Gita, almonds) ? alive(Gita)
5. ?x: ¬killed(x) ? alive(x)
6. ?x: alive(x) ? ¬killed(x)

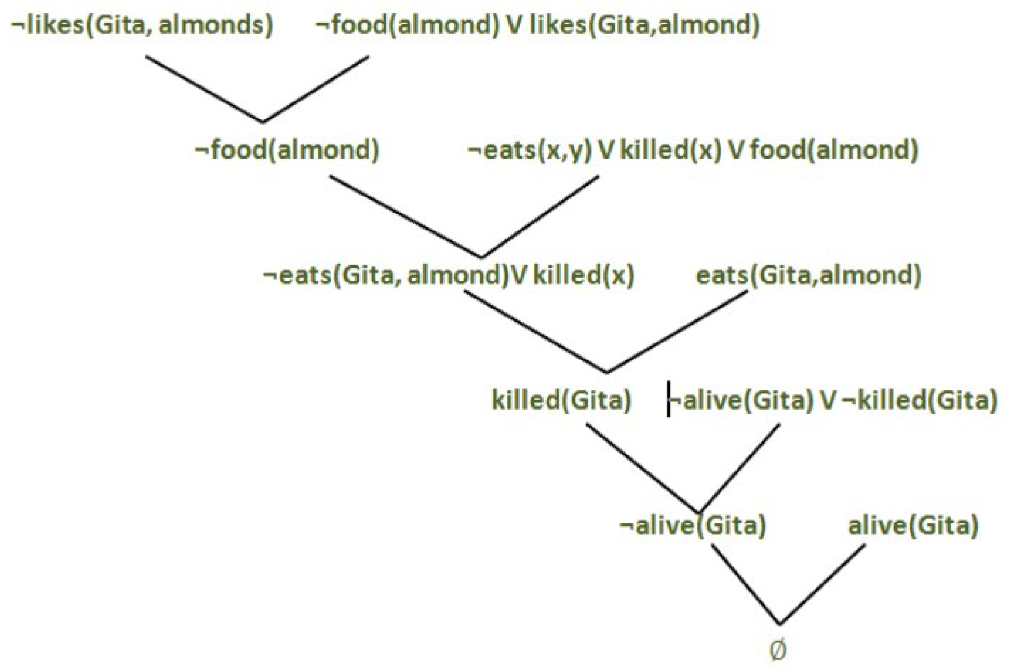
Goal: likes(Gita, almond)

Negated goal: ¬likes(Gita, almond)

Now, rewrite in CNF form:

1. ¬food(x) V likes(Gita, x)
2. food(Mango),food(chapati)
3. ¬eats(x,y) V killed(x) V food(y)
4. eats(Gita, almonds), alive(Gita)
5. killed(x) V alive(x)
6. ¬alive(x) V ¬killed(x)

Finally, construct the resolution graph:



¬likes(Gita, almonds)
¬food(almond) V likes(Gita,almond)
¬food(almond)
¬eats(x,y) V killed(x) V food(almond)
¬eats(Gita, almond)V killed(x)
eats(Gita,almond)
killed(Gita)
¬alive(Gita) V ¬killed(Gita)
¬alive(Gita)
alive(Gita)
ø

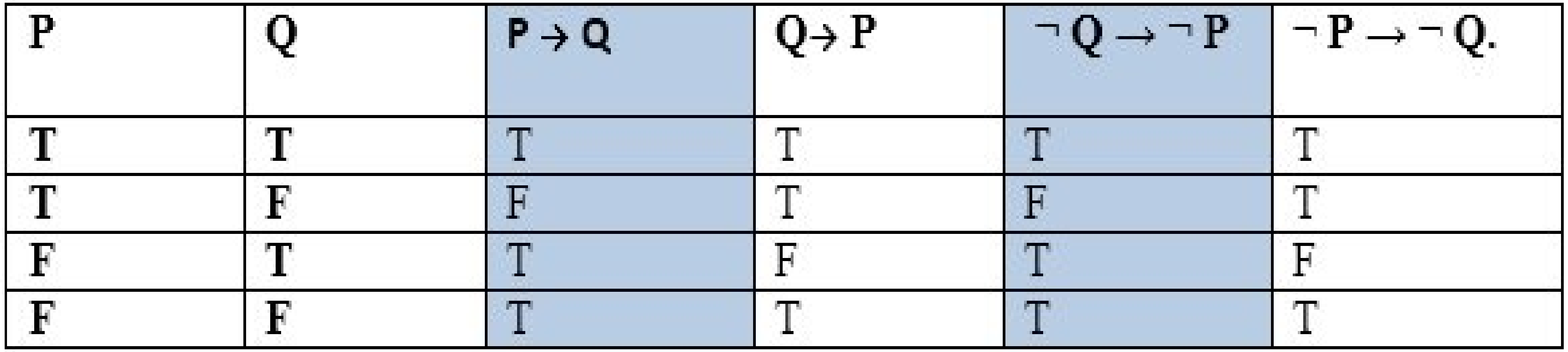
Rules of Inference in Artificial intelligence:

Inference:

Generating the conclusions from evidence and facts is termed as Inference.

Inference rules:

- **Implication:** It is one of the logical connectives which can be represented as $P \rightarrow Q$. It is a Boolean expression.
- **Converse:** The converse of implication, which means the right-hand side proposition goes to the left-hand side and vice-versa. It can be written as $Q \rightarrow P$.
- **Contrapositive:** The negation of converse is termed as contrapositive, and it can be represented as $\neg Q \rightarrow \neg P$.
- **Inverse:** The negation of implication is called inverse. It can be represented as $\neg P \rightarrow \neg Q$.



P	Q	$P \to Q$	$Q \to P$	$\neg Q \to \neg P$	$\neg P \to \neg Q$.
T	T	T	T	T	T
T	F	F	T	F	T
F	T	T	F	T	F
F	F	T	T	T	T

Hence from the above truth table, we can prove that
$P \to Q$ is equivalent to $\neg Q \to \neg P$, and
$Q \to P$ is equivalent to $\neg P \to \neg Q$.

Types of Inference rules:

1. Modus Ponens:

- The Modus Ponens rule is one of the most important rules of inference, and it states that if P and P → Q is true, then we can infer that Q will be true. It can be represented as:

Notation for Modus ponens: $$\frac{P \rightarrow Q, \quad P}{\therefore Q}$$

Example:

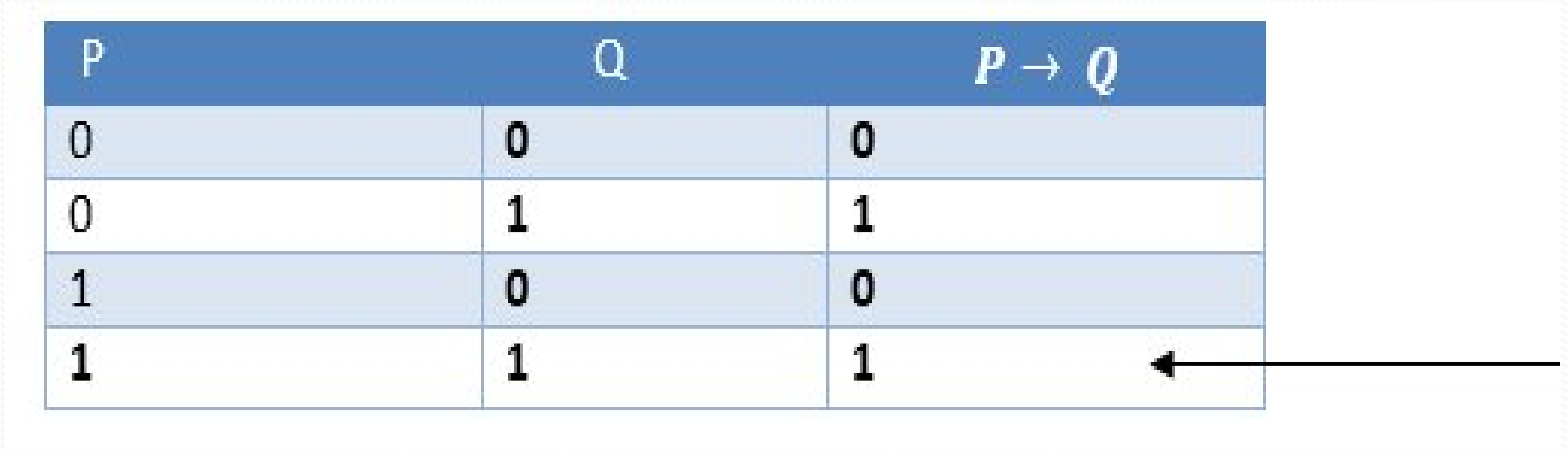
Statement-1: "If I am sleepy then I go to bed" ==> P→ Q

Statement-2: "I am sleepy" ==> P

Conclusion: "I go to bed." ==> Q.

Hence, we can say that, if P→ Q is true and P is true then Q will be true.

Proof by Truth table:



P	Q	$P \to Q$
0	0	0
0	1	1
1	0	0
1	1	1 $\leftarrow$

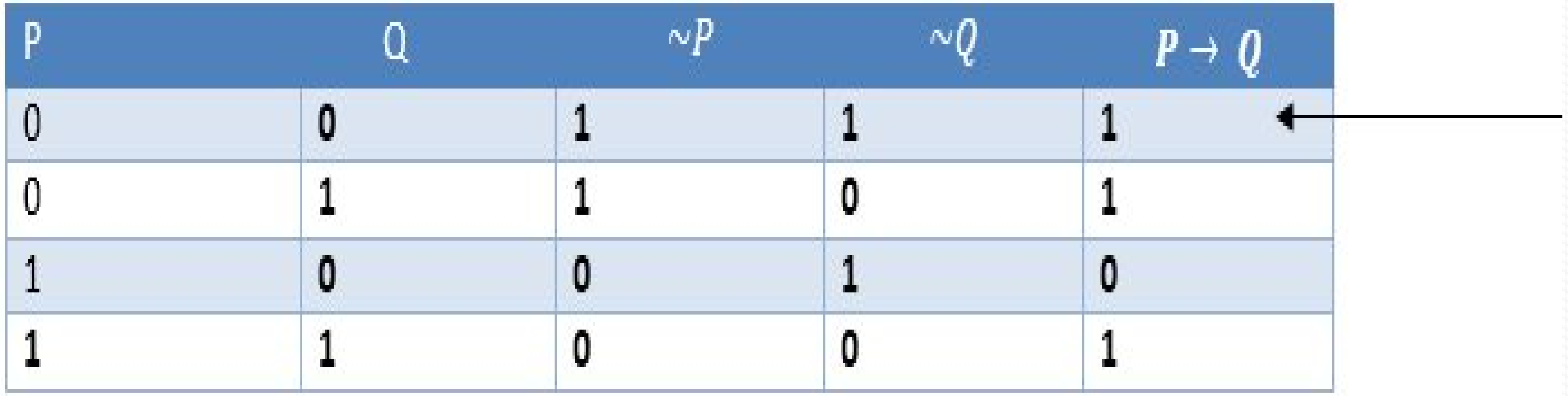
2. Modus Tollens:

- The Modus Tollens rule state that if P→ Q is true and ¬ **Q is true, then ¬ P** will also true. It can be represented as:

Notation for Modus Tollens: $$\frac{P \rightarrow Q,\ \sim Q}{\sim P}$$

Statement-1: "If I am sleepy then I go to bed" ==> P→ Q
Statement-2: "I do not go to the bed."==> ~Q
Statement-3: Which infers that "**I am not sleepy**" => ~P

Proof by Truth table:



P	Q	$\sim P$	$\sim Q$	$P \rightarrow Q$
0	0	1	1	1 ←
0	1	1	0	1
1	0	0	1	0
1	1	0	0	1

3. Hypothetical Syllogism:

- The Hypothetical Syllogism rule state that if P→R is true whenever P→Q is true, and Q→R is true. It can be represented as the following notation:

Example:

- **Statement-1:** If you have my home key then you can unlock my home. **P→Q**
  Statement-2: If you can unlock my home then you can take my money. **Q→R**
  Conclusion: If you have my home key then you can take my money. **P→R**

Proof by truth table:

P	Q	R	$P \to Q$	$Q \to R$	$P \to R$
0	0	0	1	1	1
0	0	1	1	1	1
0	1	0	1	0	1
0	1	1	1	1	1
1	0	0	0	1	1
1	0	1	0	1	1
1	1	0	1	0	0
1	1	1	1	1	1

4. Disjunctive Syllogism:

- The Disjunctive syllogism rule state that if P∨Q is true, and ¬P is true, then Q will be true. It can be represented as:

Notation of Disjunctive syllogism: $\dfrac{P \vee Q,\quad \neg P}{Q}$

Example:

Statement-1: Today is Sunday or Monday. ==>P∨Q

Statement-2: Today is not Sunday. ==> ¬P

Conclusion: Today is Monday. ==> Q

Proof by truth-table:

P	Q	$\neg P$	$P \vee Q$
0	0	1	0
0	1	1	1
1	0	0	1
1	1	0	1

5. Addition:

- The Addition rule is one the common inference rule, and it states that If P is true, then P∨Q will be true.

Notation of Addition: $\dfrac{P}{P \vee Q}$

Example:
Statement: I have a vanilla ice-cream. ==> P
Statement-2: I have Chocolate ice-cream.
Conclusion: I have vanilla or chocolate ice-cream. ==> (P∨Q)

Proof by Truth-Table:

P	Q	$P \vee Q$	
0	0	0	
1	0	1	←
0	1	1	
1	1	1	←

6. Simplification:

- The simplification rule state that if **P∧ Q** is true, then **Q or P** will also be true. It can be represented as:

Notation of Simplification rule: $\dfrac{P \land Q}{Q}$ Or $\dfrac{P \land Q}{P}$

- **Proof by Truth-Table:**

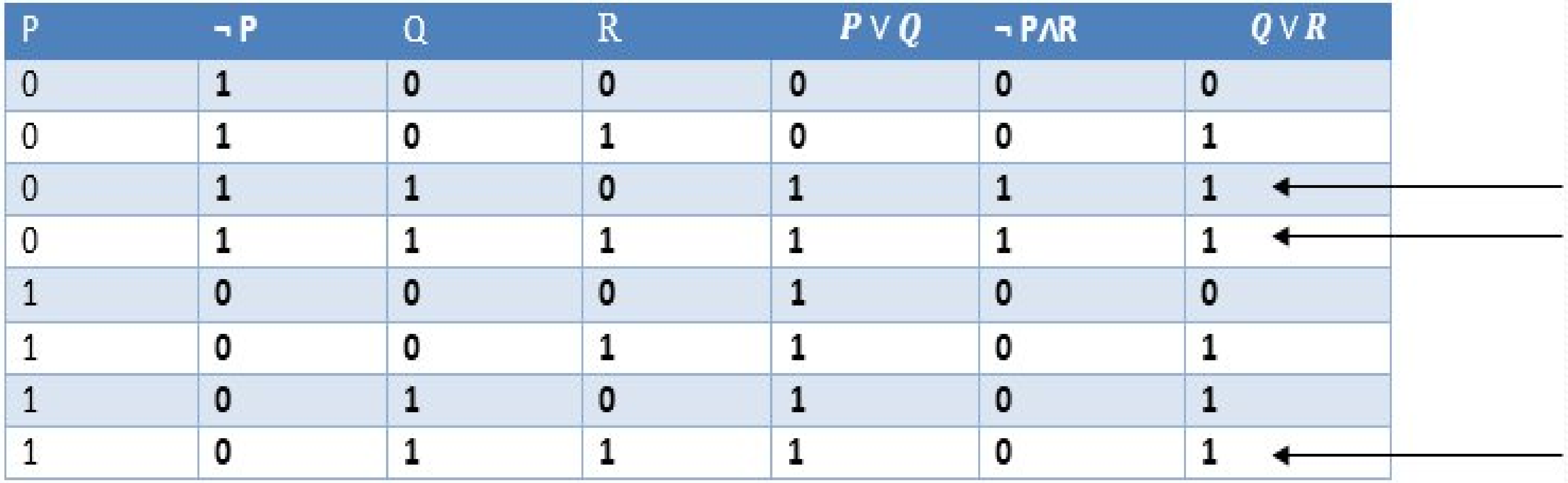
P	Q	$P \land Q$
0	0	0
1	0	0
0	1	0
1	1	1 ←

7. Resolution:

- The Resolution rule state that if P∨Q and ¬ P∧R is true, then Q∨R will also be true. **It can be represented as**

Notation of Resolution $\frac{P \vee Q,\ \neg P \wedge R}{Q \vee R}$

Proof by Truth-Table:



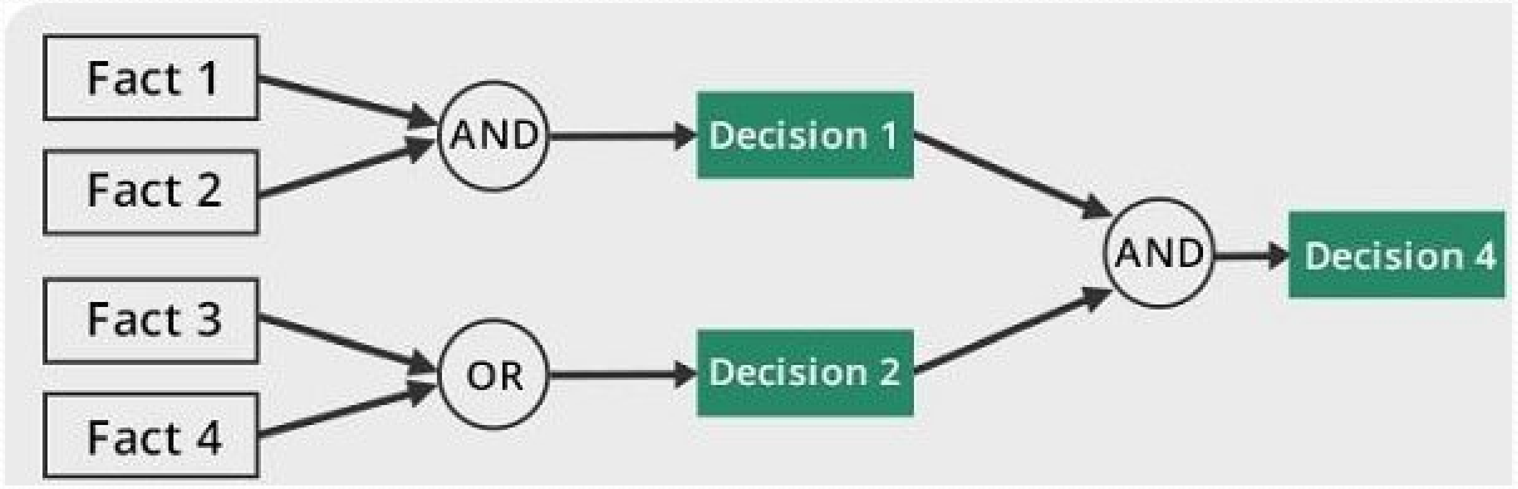
P	¬P	Q	R	$P \vee Q$	¬P∧R	$Q \vee R$	
0	1	0	0	0	0	0	
0	1	0	1	0	0	1	
0	1	1	0	1	1	1	←
0	1	1	1	1	1	1	←
1	0	0	0	1	0	0	
1	0	0	1	1	0	1	
1	0	1	0	1	0	1	
1	0	1	1	1	0	1	←

Forward and Backward Reasoning

- Backward and forward chaining stem from the inference engine component.
- This is a component in which logical rules are applied to the knowledge base to get new information or make a decision.
- The backward and forward chaining techniques are used by the inference engine as strategies for proposing solutions or deducing information in the expert system.

Forward reasoning:

- Forward chaining is a method of reasoning in artificial intelligence in which inference rules are applied to existing data to extract additional data until an endpoint (goal) is achieved.
- In this type of chaining, the inference engine starts by evaluating existing facts, derivations, and conditions before deducing new information. An endpoint (goal) is achieved through the manipulation of knowledge that exists in the knowledge base.



Properties of forward chaining

- The process uses a down-up approach (bottom to top).
- It starts from an initial state and uses facts to make a conclusion.
- This approach is data-driven.
- It's employed in expert systems and production rule system.

Example:

A

A->B

B

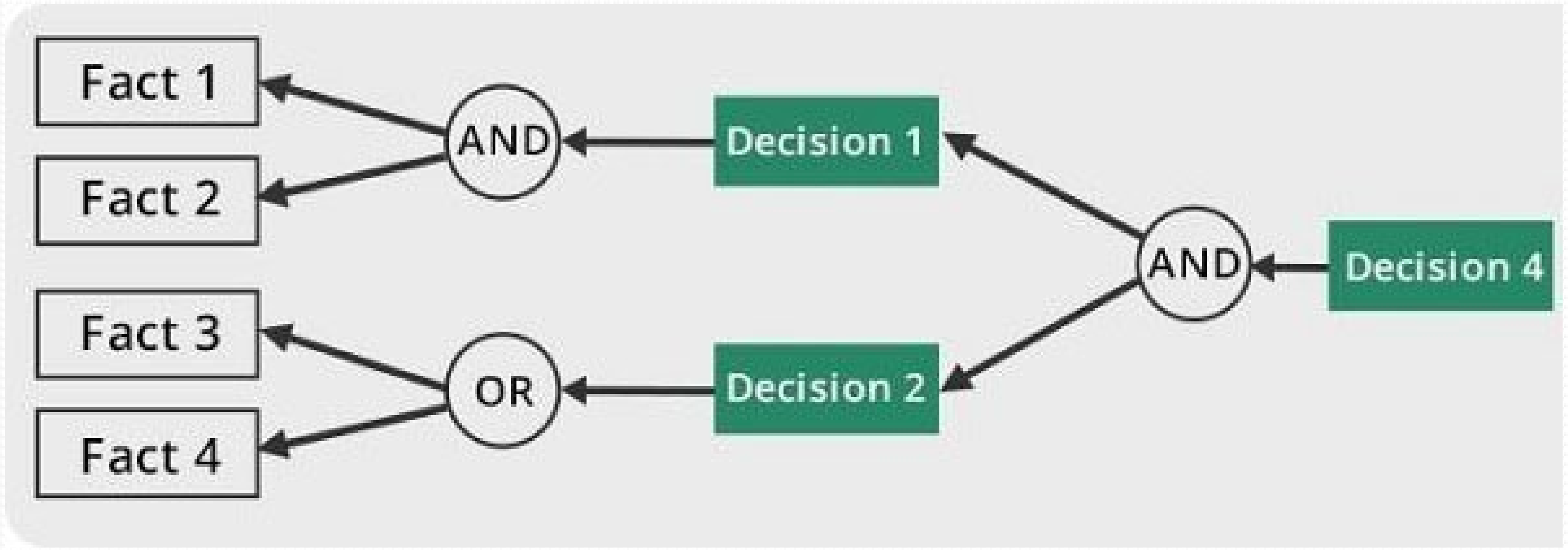
A is the starting point. A->B represents a fact. This fact is used to achieve a decision B.

A practical example will go as follows;

- Tom is running (A)
- If a person is running, he will sweat (A->B)
- Therefore, Tom is sweating. (B)

Backward chaining:

- Backward chaining is a concept in artificial intelligence that involves backtracking from the endpoint or goal to steps that led to the endpoint. This type of chaining starts from the goal and moves backward to comprehend the steps that were taken to attain this goal.
- The backtracking process can also enable a person establish logical steps that can be used to find other important solutions.



- Backward chaining can be used in debugging, diagnostics, and prescription applications.

Properties of backward chaining

- The process uses an up-down approach (top to bottom).
- It’s a goal-driven method of reasoning.
- The endpoint (goal) is subdivided into sub-goals to prove the truth of facts.
- A backward chaining algorithm is employed in inference engines, game theories, and complex database systems.
- The modus ponens inference rule is used as the basis for the backward chaining process. This rule states that if both the conditional statement (p->q) and the antecedent (p) are true, then we can infer the subsequent (q).

Example of backward chaining:

- The information provided in the previous example (forward chaining) can be used to provide a simple explanation of backward chaining. Backward chaining can be explained in the following sequence.

  B

  A->B

  A

- B is the goal or endpoint, that is used as the starting point for backward tracking. A is the initial state. A->B is a fact that must be asserted to arrive at the endpoint B.

A practical example of backward chaining will go as follows:

- Tom is sweating (B).
- If a person is running, he will sweat (A->B).
- Tom is running (A).