ARTIFICIAL INTELLIGENCE

	L	T	P	C
BCSE306L -	3	0	0	3

VIT®

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

Dr. S M SATAPATHY

Associate Professor Sr.,
School of Computer Science and Engineering,
VIT Vellore, TN, India – 632 014.

Module – 4

LOGIC & REASONING

1. Logical systems
2. Knowledge Based systems
3. Propositional Logic Constraints
4. Predicate Logic
5. First Order Logic
6. Inference in First Order Logic

KNOWLEDGE BASED SYSTEMS

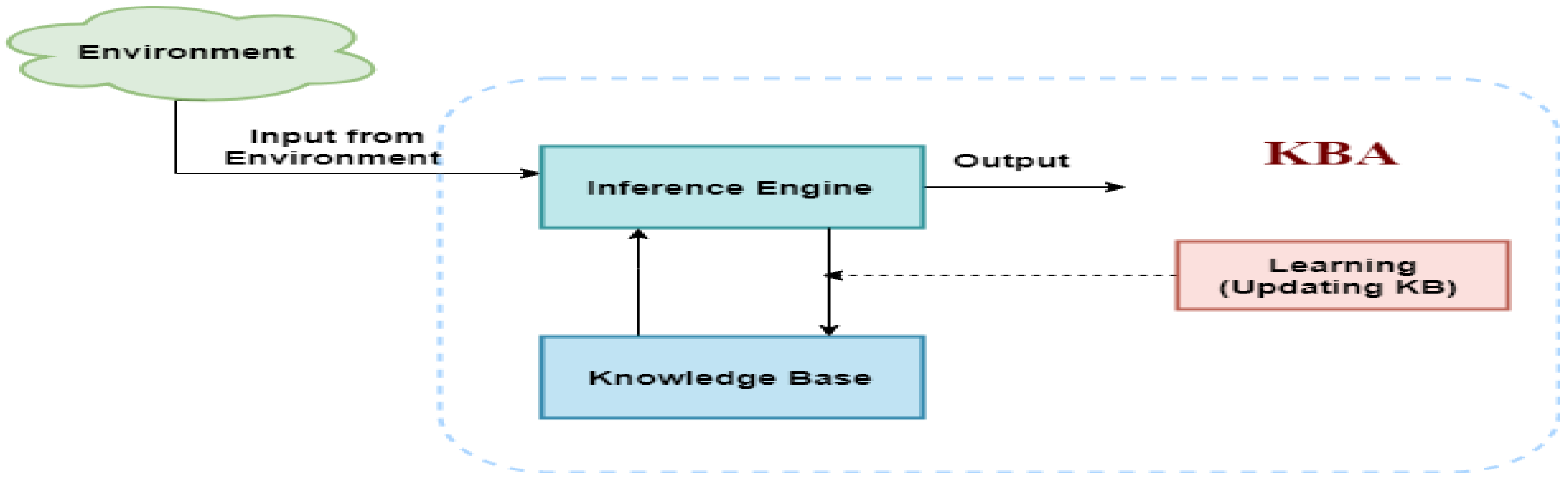
Knowledge based Agent :: Definition

- An intelligent agent needs knowledge about the real world for taking decisions and reasoning to act efficiently.
- 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. These agents can represent the world with some formal representation and act intelligently. Inference engine is a component of the system that applies logical rules to the knowledge base to deduce new information
- Knowledge-based agents are composed of two main parts:
  - Knowledge-base and
  - Inference system.

Knowledge based Agent :: Properties

- An agent should be able to represent states, actions, etc.
- An agent should be able to incorporate new precepts.
- An agent can update the internal representation of the world.
- An agent can deduce the internal representation of the world.
- An agent can deduce appropriate actions.

Knowledge based Agent :: Architecture



❖ The above diagram is representing a generalized architecture for a knowledge-based agent.

❖ The knowledge-based agent (KBA) take input from the environment by perceiving the environment.

❖ The input is taken by the inference engine of the agent and which also communicate with KB to decide as per the knowledge store in KB.

❖ The learning element of KBA regularly updates the KB by learning new knowledge.

Why use a knowledge base?

- Knowledge-base is required for updating knowledge for an agent to learn with experiences and take action as per the knowledge.
- Knowledge-base is a central component of a knowledge-based agent, it is also known as KB.
- It is a collection of sentences (here 'sentence' is a technical term and it is not identical to sentence in English).
- These sentences are expressed in a language which is called a knowledge representation language.
- The Knowledge-base of KBA stores fact about the world.

Knowledge Base :: Architecture

Knowledge level

- ❖ Knowledge level is the first level of knowledge-based agent, and in this level, we need to specify what the agent knows, and what the agent goals are. With these specifications, we can fix its behavior. For example, suppose an automated taxi agent needs to go from a station A to station B, and he knows the way from A to B, so this comes at the knowledge level.

Logical level

- ❖ At this level, we understand that how the knowledge representation of knowledge is stored. At this level, sentences are encoded into different logics. At the logical level, an encoding of knowledge into logical sentences occurs. At the logical level we can expect to the automated taxi agent to reach to the destination B.

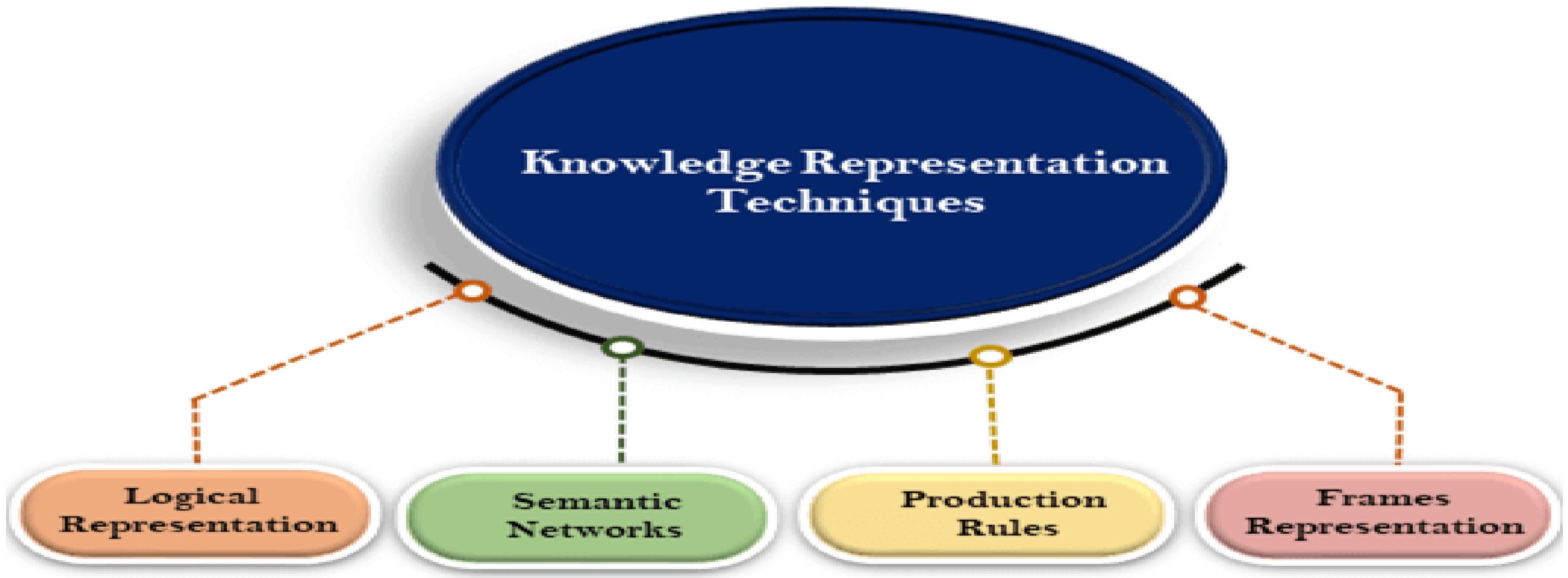
Knowledge Base :: Architecture

Implementation level

- This is the physical representation of logic and knowledge. At the implementation level agent perform actions as per logical and knowledge level. At this level, an automated taxi agent actually implement his knowledge and logic so that he can reach to the destination.

Knowledge Representation Techniques

- There are mainly four ways of knowledge representation which are given as follows:
  - Logical Representation
  - Semantic Network Representation
  - Frame Representation
  - Production Rules



Techniques :: Logical Representation

- Logical representation means drawing a conclusion based on various conditions.
- This representation lays down some important communication rules.
- It consists of precisely defined syntax and semantics which supports the sound inference.
- Each sentence can be translated into logics using syntax and semantics.

Techniques :: Logical Representation

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 categorized into mainly two logics
  - Propositional Logics
  - Predicate logics

Techniques :: Logical Representation – Adv. & Disadv.

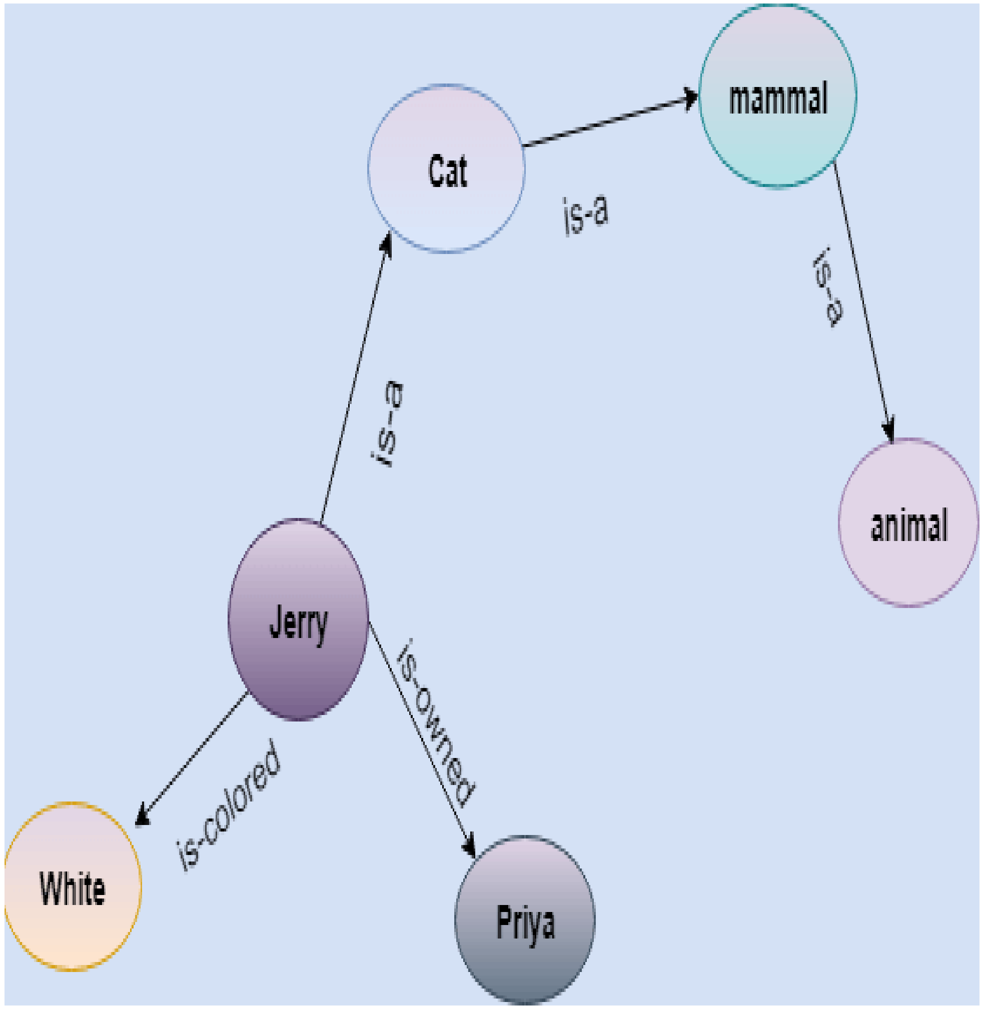
- Logical representation enables us to do logical reasoning.
- Logical representation is the basis for the programming languages.
- Logical representations have some restrictions and are challenging to work with.
- Logical representation technique may not be very natural, and inference may not be so efficient.

Techniques :: Semantic Network Representation

- Semantic networks 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.
- Semantic networks are easy to understand and can be easily extended.
- This representation consist of mainly two types of relations:
    - IS-A relation (Inheritance)
    - Kind-of-relation

Techniques :: Semantic Network Representation

- Jerry is a cat.
- Jerry is a mammal
- Jerry is owned by Priya.
- Jerry is white colored.
- All Mammals are animal.



Techniques :: Semantic Network Representation - Drawbacks

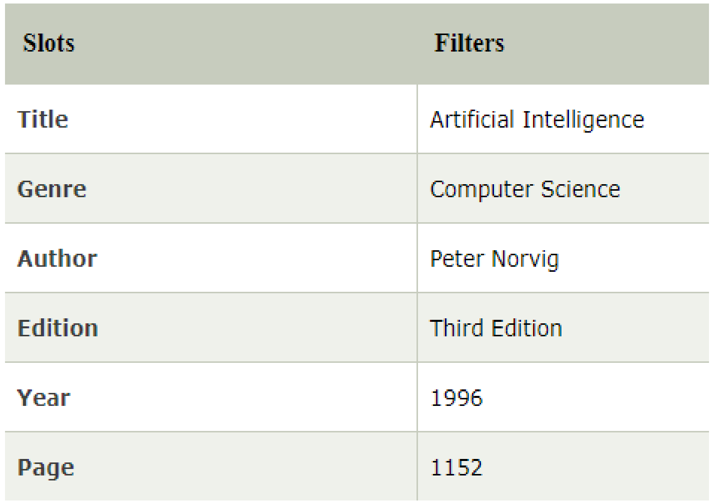
- Semantic networks take more computational time at runtime as we need to traverse the complete network tree to answer some questions. It might be possible in the worst case scenario that after traversing the entire tree, we find that the solution does not exist in this network.
- Semantic networks do not have any standard definition for the link names.
- These networks are not intelligent and depend on the creator of the system.

Techniques :: Semantic Network Representation - Advantages

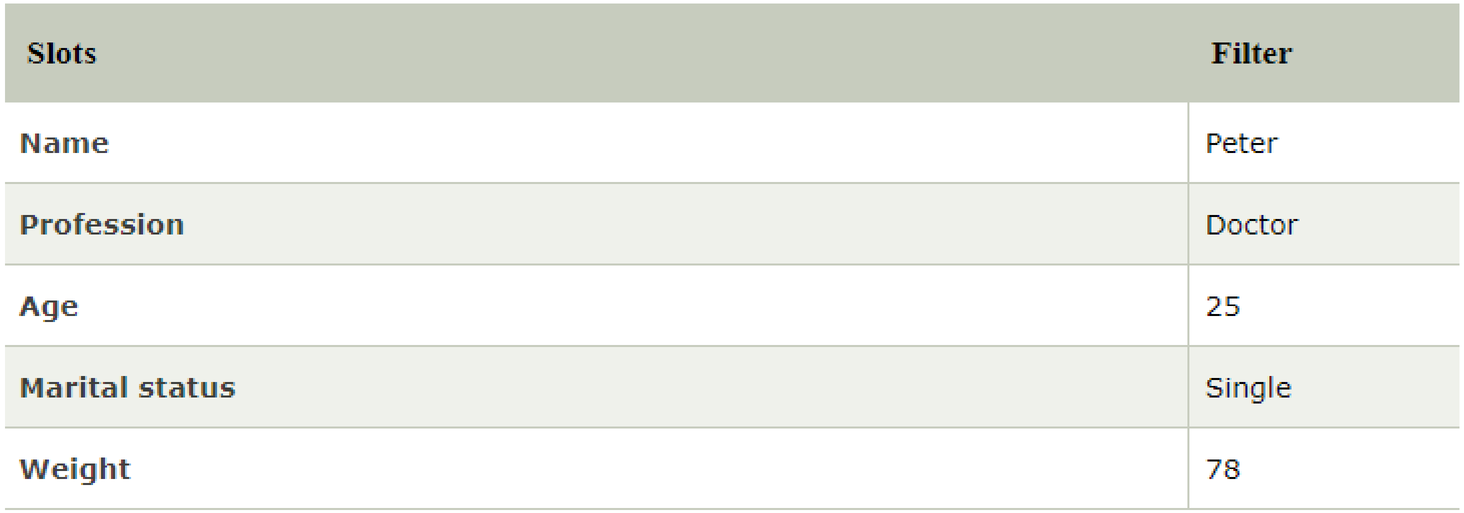
- Semantic networks are a natural representation of knowledge.
- Semantic networks convey meaning in a transparent manner.
- These networks are simple and easily understandable.

Techniques :: 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.



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



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

Techniques :: Production Rule

- Production rules system consist of (condition, action) pairs which mean, "If condition then action“.
- 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).

Techniques :: Production Rule – Adv. & Disadv.

- The production rules are expressed in natural language.
- The production rules are highly modular, so we can easily remove, add or modify an individual rule.
- Production rule system does not exhibit any learning capabilities, as it does not store the result of the problem for the future uses.
- During the execution of the program, many rules may be active hence rule-based production systems are inefficient.

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.
- It is a technique of knowledge representation in logical and mathematical form.

Example

- It is Sunday.
- The Sun rises from West (False proposition)
- 3+3= 7(False proposition)
- 5 is a prime number. (True Proposition)

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 representing a proposition, such A, B, C, P, Q, R, etc.
- Propositions can be either true or false, but it cannot be both.
- Propositional logic consists of an object, relations or function, and logical connectives. These connectives are also called logical operators.
- The propositions and connectives are the basic elements of the propositional logic.
- Connectives can be said as a logical operator which connects two sentences.

Propositional Logic

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

Propositional Logic

- There are two types of Propositions
  - Atomic Propositions
  - Compound propositions

Atomic Proposition

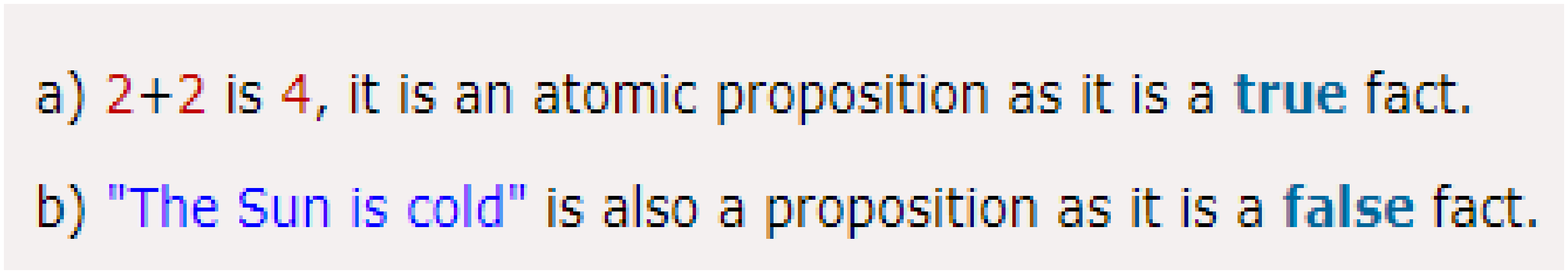
- Atomic propositions are the simple propositions. It consists of a single proposition symbol. These are the sentences which must be either true or false.

Compound proposition

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

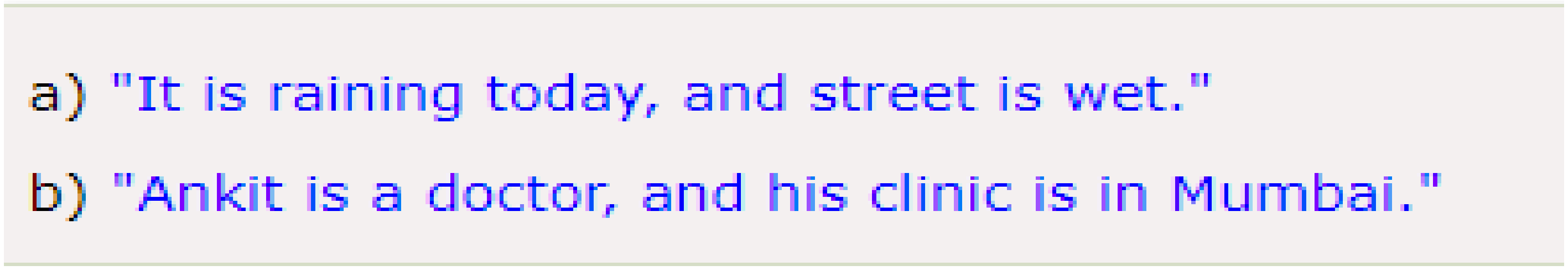
Propositional Logic

❖ 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.



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:
- ❖ Negation: A sentence such as $\neg$ P is called negation of P. A literal can be either Positive literal or negative literal.
- ❖ Conjunction: A sentence which has $\wedge$ connective such as, $P \wedge Q$ is called a conjunction.

  Example: Rohan is intelligent and hardworking. It can be written as,

  P= Rohan is intelligent,

  Q= Rohan is hardworking. $\rightarrow P\wedge Q$.

Logical Connectives

- Disjunction: A sentence which has $\vee$ connective, such as $P \vee Q$. 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 Engineer,

  so we can write it as $P \vee Q$.

- Implication: A sentence such as $P \rightarrow Q$, is called an implication. Implications are also known as if-then rules. It can be represented as

  If it is raining, then the street is wet.

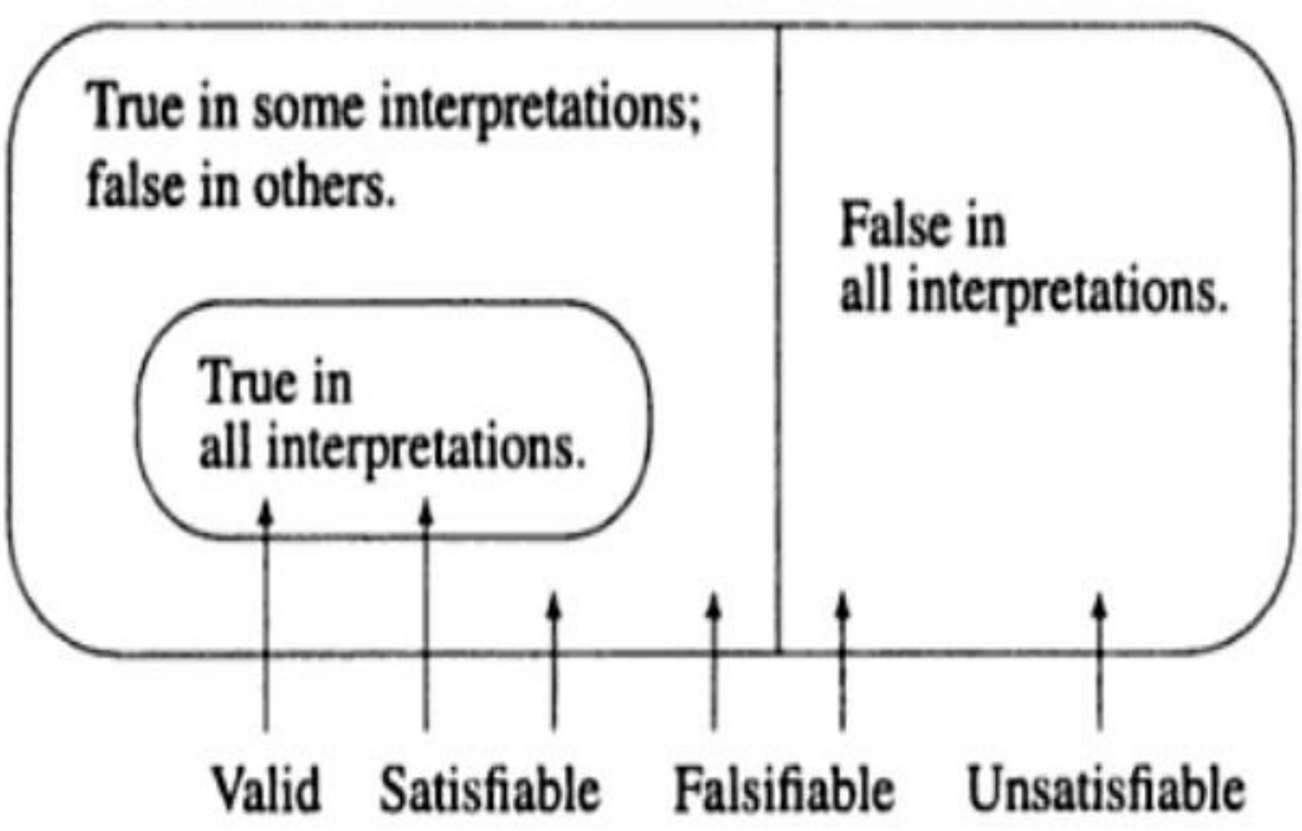
  Let P= It is raining, and Q= Street is wet, so it is represented as $P \rightarrow Q$

- Biconditional: A sentence such as $P \Leftrightarrow Q$ is a Biconditional sentence, example If I am breathing, then I am alive

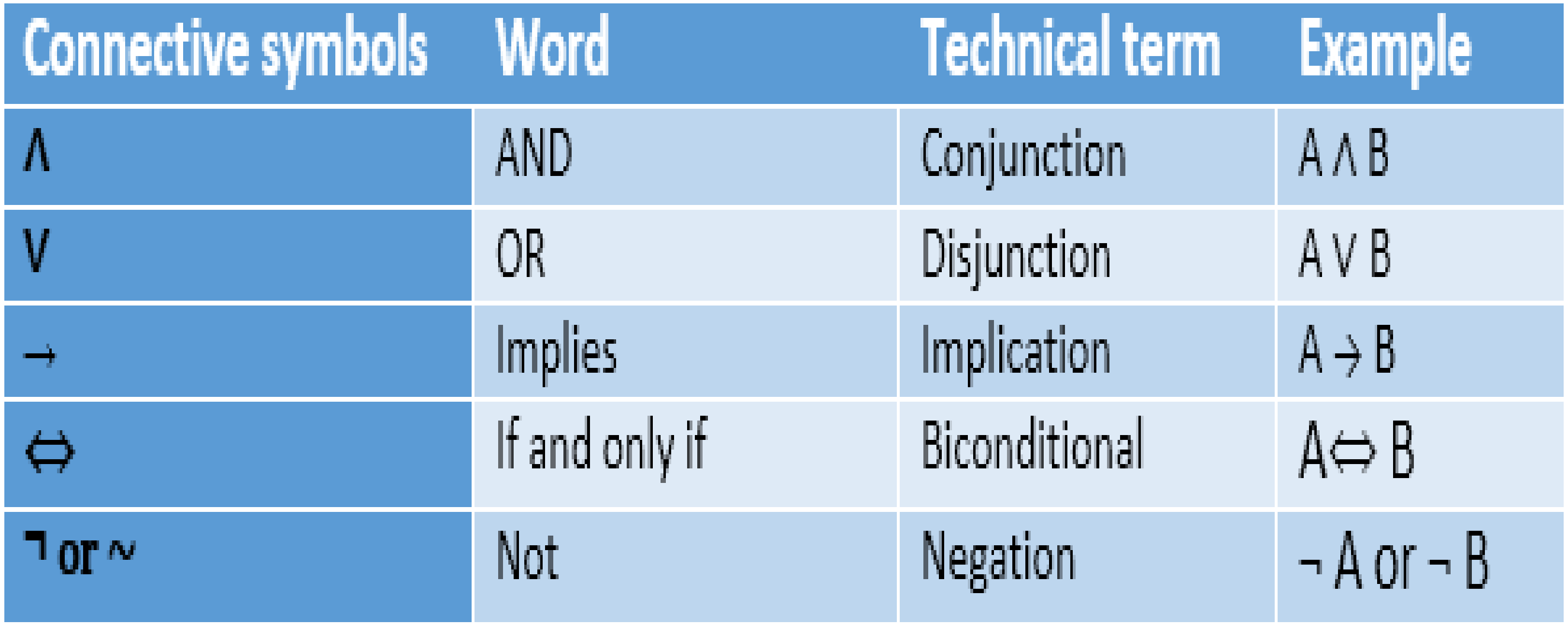
  P= I am breathing, Q= I am alive, it can be represented as $P \Leftrightarrow Q$.

Semantics

- Valid Statements
- Inconsistent Statements/ Contradiction
- P entails Q written P ⊨ Q, means that whenever P is True, so is Q.

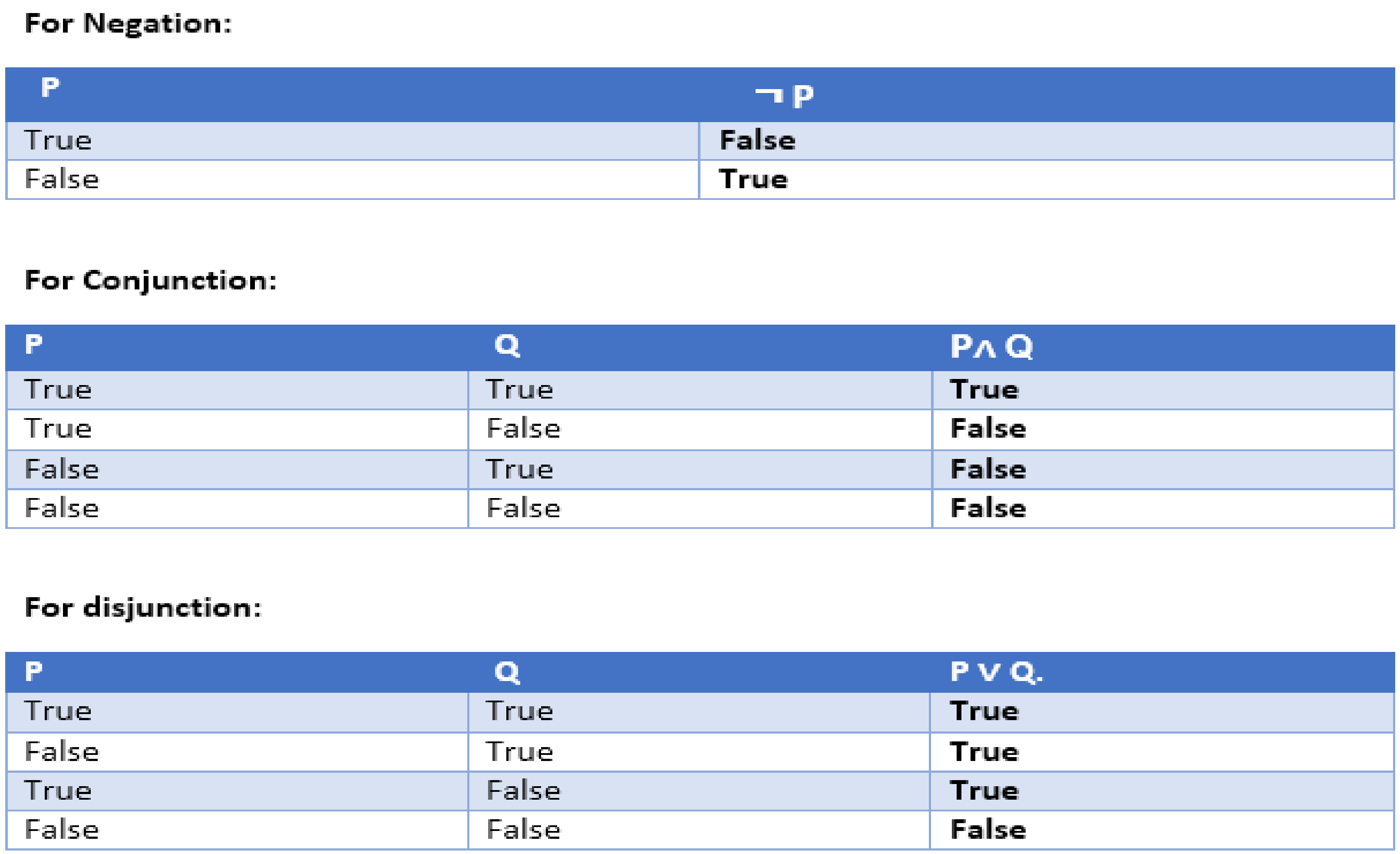

True in some interpretations; false in others.
True in all interpretations.
False in all interpretations.
Valid
Satisfiable
Falsifiable
Unsatisfiable

Propositional Logic Connectives



Connective symbols	Word	Technical term	Example
$\wedge$	AND	Conjunction	$A \wedge B$
$\vee$	OR	Disjunction	$A \vee B$
$\rightarrow$	Implies	Implication	$A \rightarrow B$
$\Leftrightarrow$	If and only if	Biconditional	$A \Leftrightarrow B$
$\neg$ or $\sim$	Not	Negation	$\neg A$ or $\neg B$

Propositional Logic Connectives :: Truth Table



For Negation:

P	¬ P
True	**False**
False	**True**

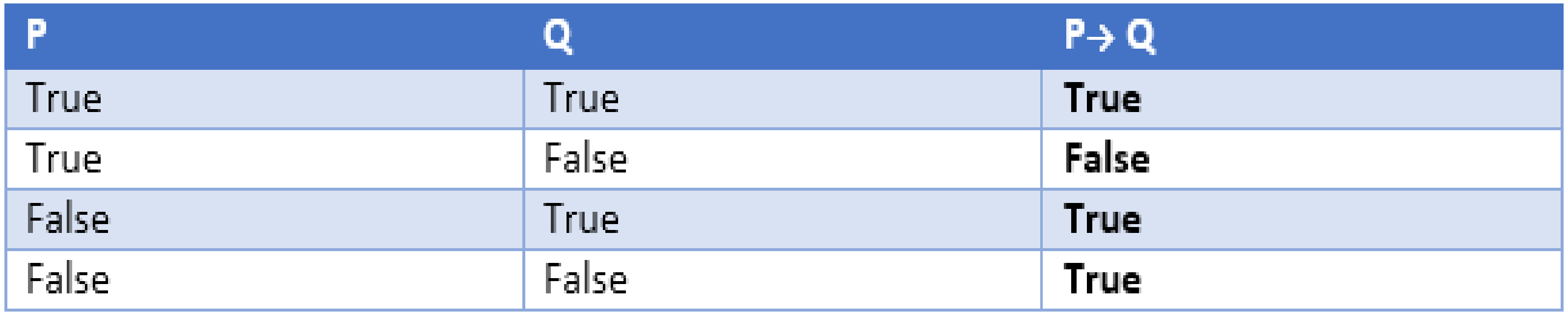
For Conjunction:

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

For disjunction:

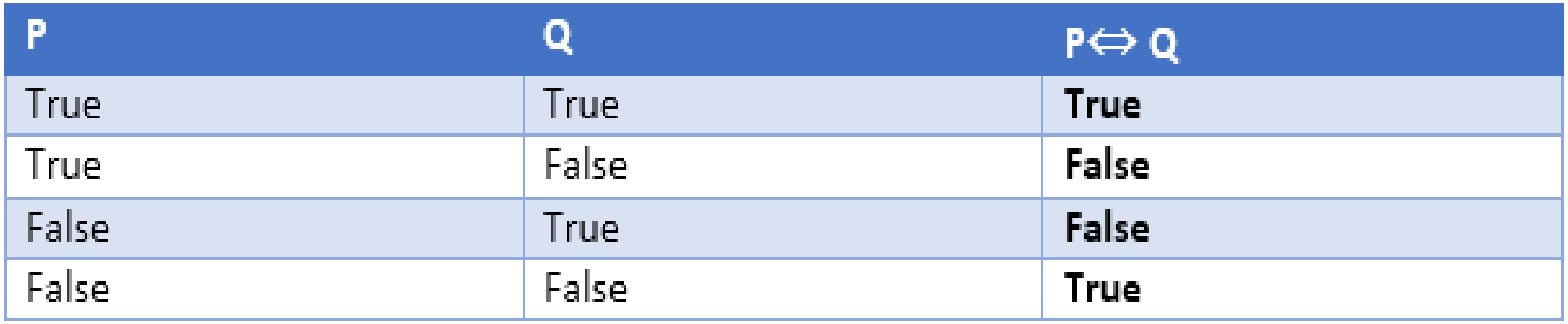
P	Q	P ∨ Q.
True	True	**True**
False	True	**True**
True	False	**True**
False	False	**False**

Propositional Logic Connectives :: Truth Table



For Implication:

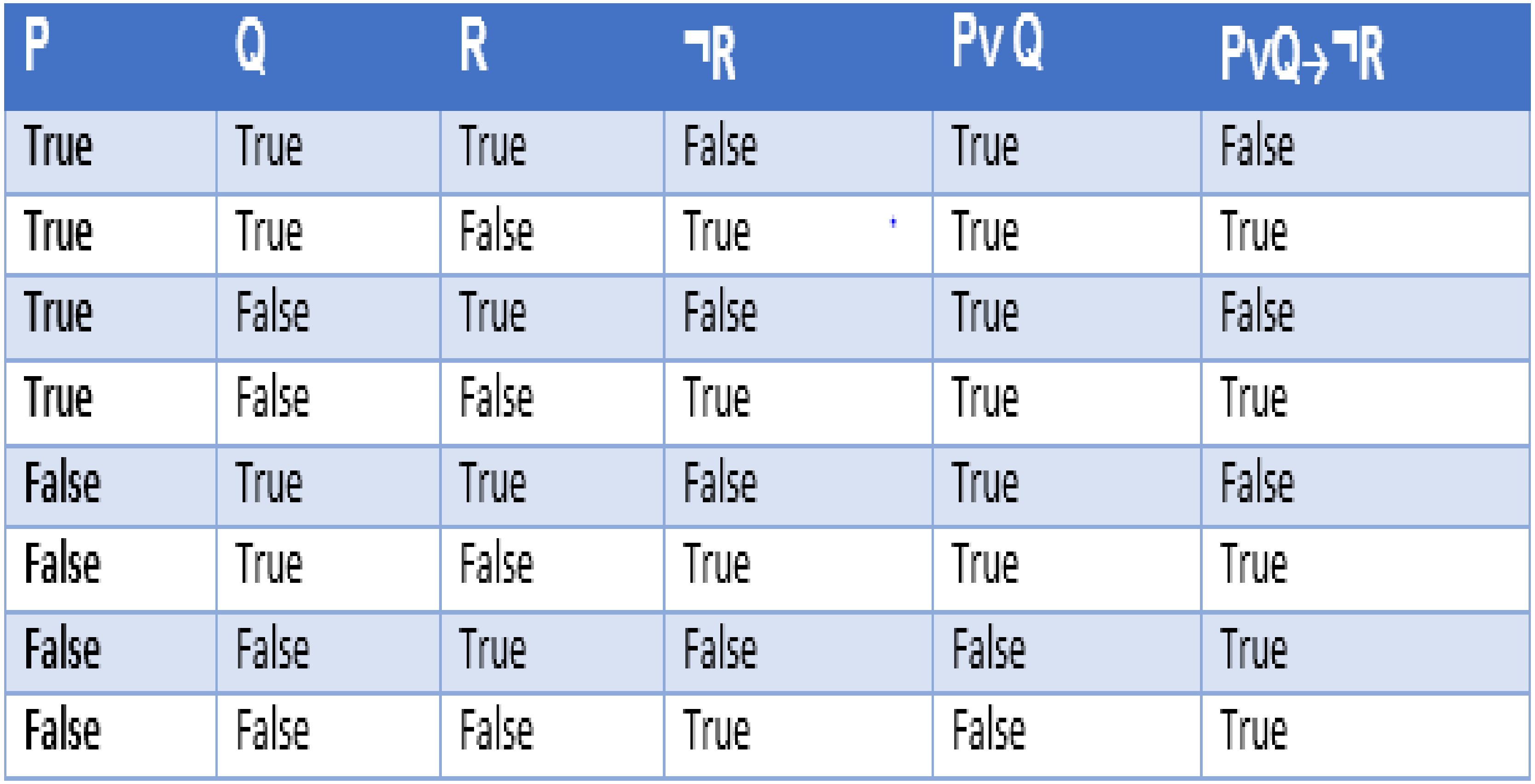
P	Q	P→ Q
True	True	**True**
True	False	**False**
False	True	**True**
False	False	**True**



For Biconditional:

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

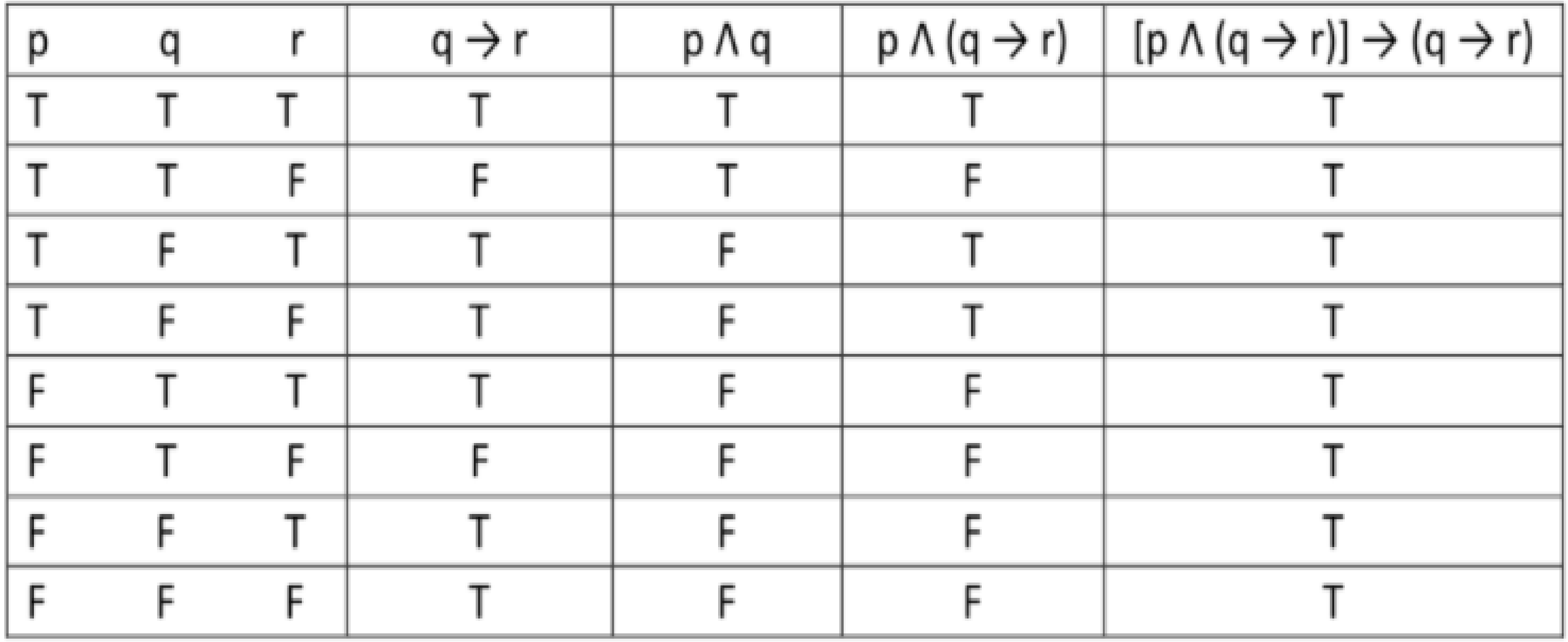
Truth table with three propositions



P	Q	R	$\neg R$	$P \vee Q$	$P \vee Q \rightarrow \neg R$
True	True	True	False	True	False
True	True	False	True	True	True
True	False	True	False	True	False
True	False	False	True	True	True
False	True	True	False	True	False
False	True	False	True	True	True
False	False	True	False	False	True
False	False	False	True	False	True

Example

$$[p \wedge (q \rightarrow r)] \rightarrow (q \rightarrow r)$$



p	q	r	$q \rightarrow r$	$p \wedge q$	$p \wedge (q \rightarrow r)$	$[p \wedge (q \rightarrow r)] \rightarrow (q \rightarrow r)$
T	T	T	T	T	T	T
T	T	F	F	T	F	T
T	F	T	T	F	T	T
T	F	F	T	F	T	T
F	T	T	T	F	F	T
F	T	F	F	F	F	T
F	F	T	T	F	F	T
F	F	F	T	F	F	T

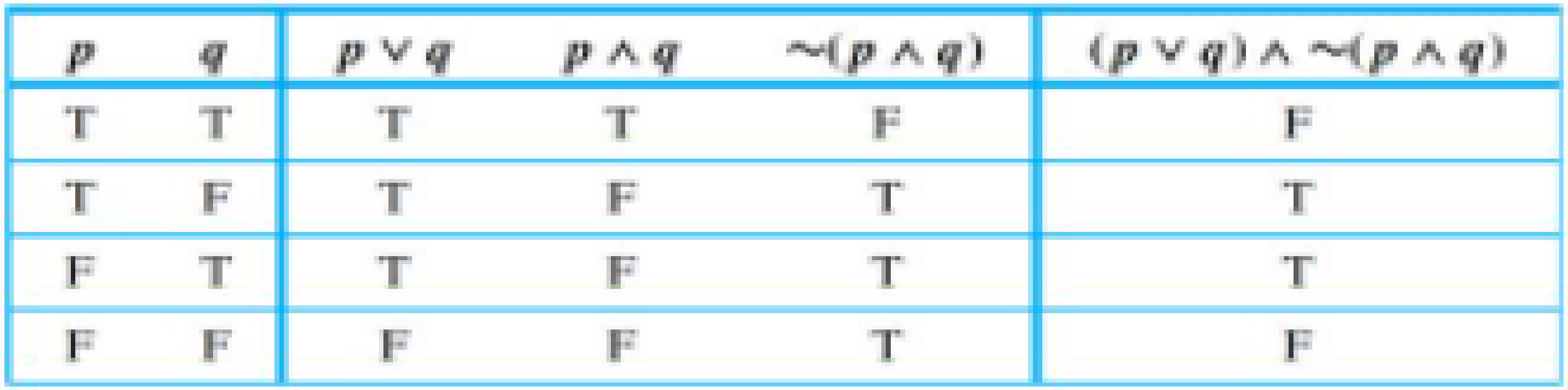
Example

Set up columns labeled p, q, $p \vee q$, $p \wedge q$, $\sim(p \wedge q)$, and $(p \vee q) \wedge \sim(p \wedge q)$.

$$(p \vee \neg q) \rightarrow (p \wedge q).$$

$$(A \rightarrow B) \vee (B \rightarrow C)$$

Example



p	q	$p \vee q$	$p \wedge q$	$\sim(p \wedge q)$	$(p \vee q) \wedge \sim(p \wedge q)$
T	T	T	T	F	F
T	F	T	F	T	T
F	T	T	F	T	T
F	F	F	F	T	F

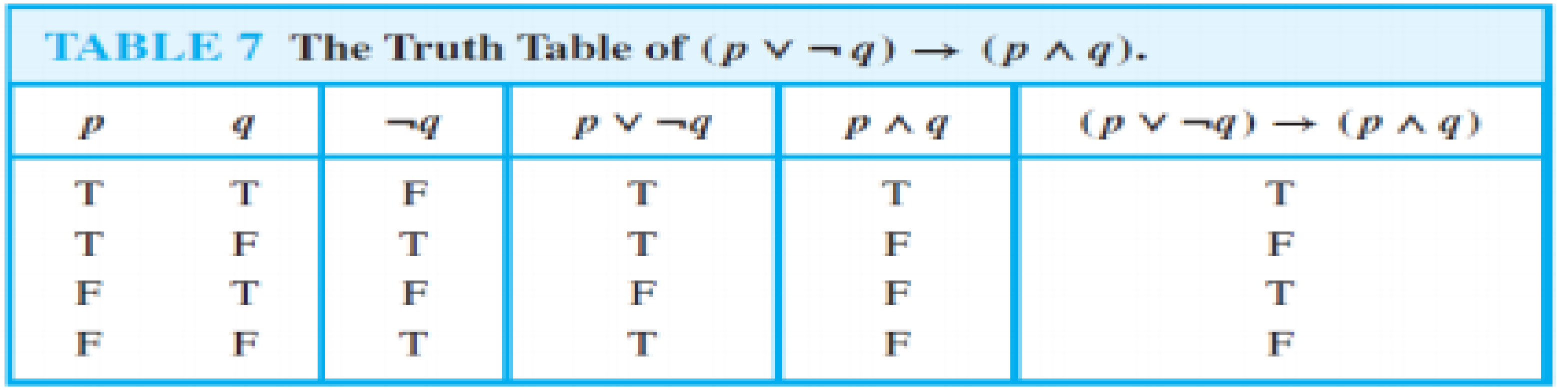
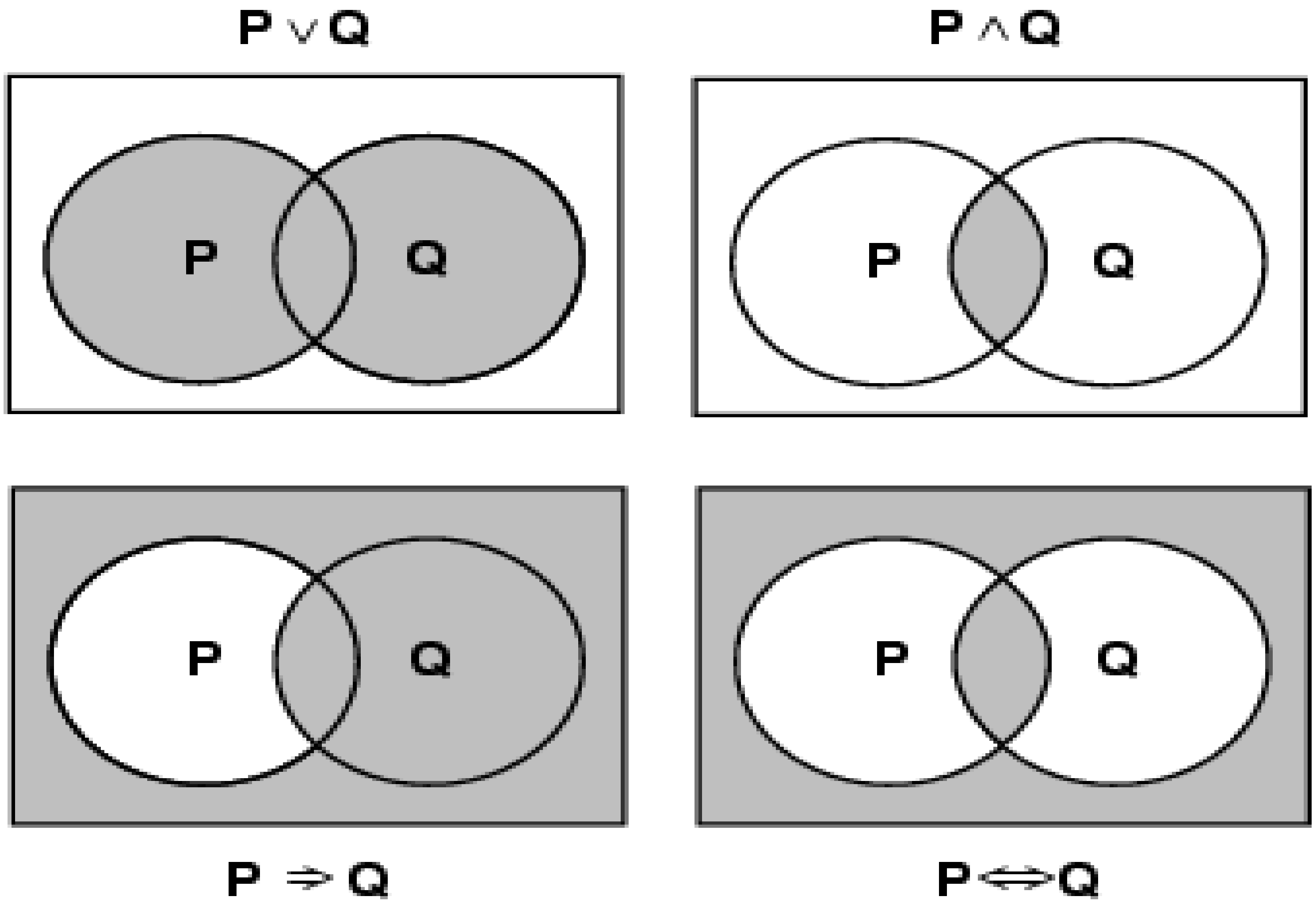


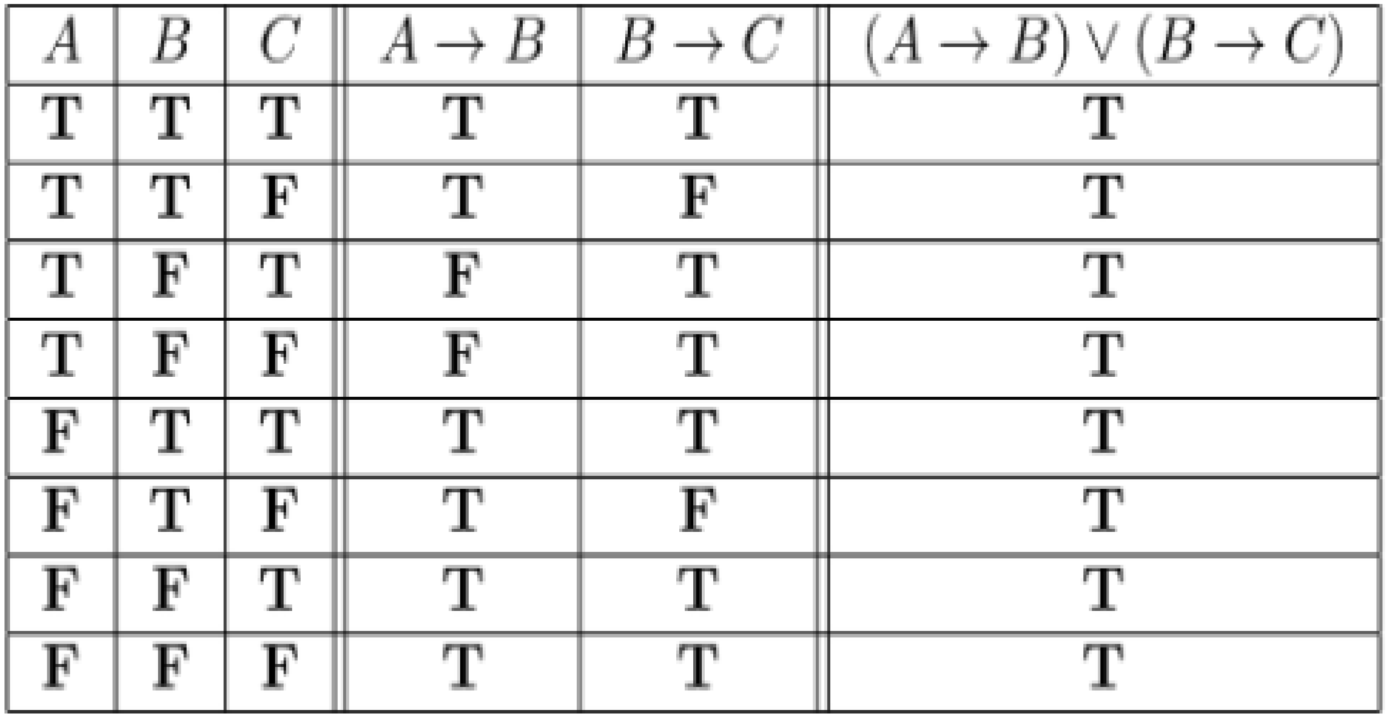
TABLE 7 The Truth Table of $(p \vee \neg q) \rightarrow (p \wedge q)$.

p	q	$\neg q$	$p \vee \neg q$	$p \wedge q$	$(p \vee \neg q) \rightarrow (p \wedge q)$
T	T	F	T	T	T
T	F	T	T	F	F
F	T	F	F	F	T
F	F	T	T	F	F

Models of complex sentences



Example



A	B	C	$A \to B$	$B \to C$	$(A \to B) \vee (B \to C)$
T	T	T	T	T	T
T	T	F	T	F	T
T	F	T	F	T	T
T	F	F	F	T	T
F	T	T	T	T	T
F	T	F	T	F	T
F	F	T	T	T	T
F	F	F	T	T	T

Example

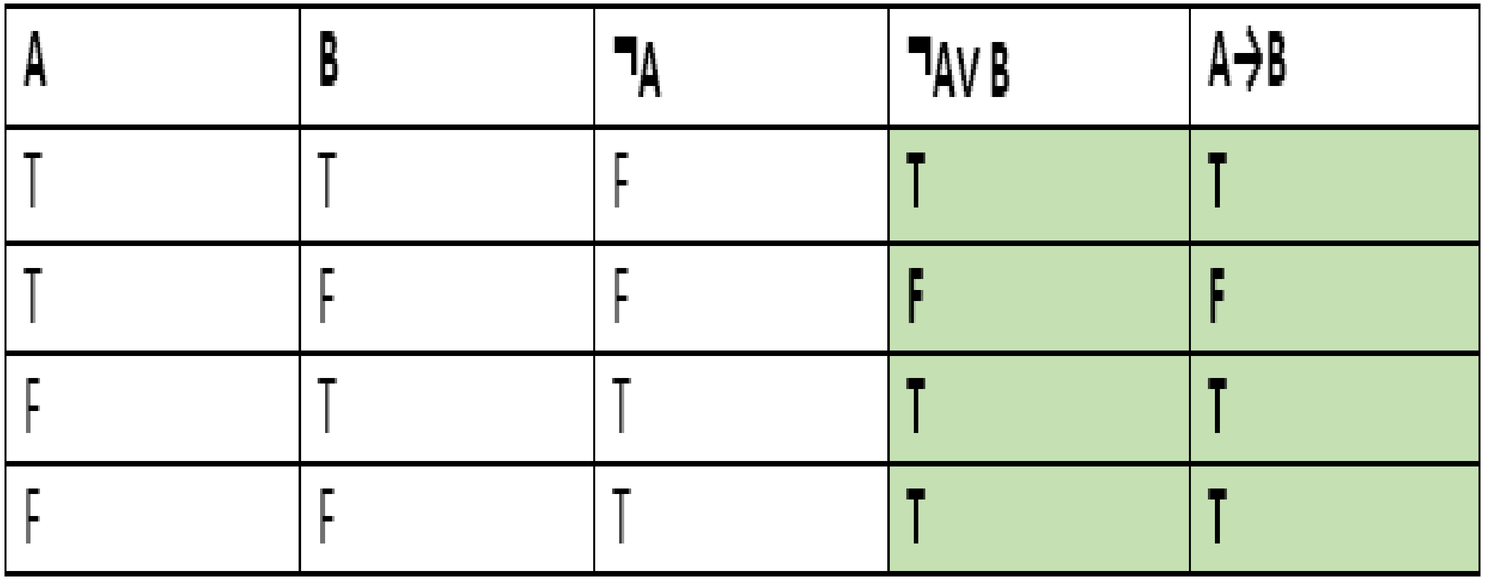
Show that $A \to B$, $\neg A \vee B$, and $\neg(A \wedge \neg B)$ are logically equivalent, by writing out the truth table and showing that they have the same values for all truth assignments.

Show that $(p \to q) \wedge (q \to p)$ is logically equivalent to $p \leftrightarrow q$.

- Valid (tautology): $(p \to q) \to (\neg q \to \neg p)$
- Not valid, but satisfiable: $q \to (q \to p)$
- False (contradiction): $(p \wedge \neg p) \vee (q \wedge \neg q)$

Logical equivalence

- Logical equivalence is one of the features of propositional logic. Two propositions are said to be logically equivalent if and only if the columns in the truth table are identical to each other.
- Let's take two propositions A and B, so for logical equivalence, we can write it as A⇔B. In below truth table we can see that column for ¬A∨ B and A→B, are identical hence A is Equivalent to B



A	B	¬A	¬A∨ B	A→B
T	T	F	T	T
T	F	F	F	F
F	T	T	T	T
F	F	T	T	T

Properties of Operators

Commutative:

- P∧ Q= Q ∧ P, or
- P ∨ Q = Q ∨ P.

Associative:

- (P ∧ Q) ∧ R= P ∧ (Q ∧ R),
- (P ∨ Q) ∨ R= P ∨ (Q ∨ R)

Identity element:

- P ∧ True = P,
- P ∨ True= True.

Distributive:

- P∧ (Q ∨ R) = (P ∧ Q) ∨ (P ∧ R).
- P ∨ (Q ∧ R) = (P ∨ Q) ∧ (P ∨ R).

DE Morgan's Law:

- ¬ (P ∧ Q) = (¬P) ∨ (¬Q)
- ¬ (P ∨ Q) = (¬ P) ∧ (¬Q).

Double-negation elimination:

- ¬ (¬P) = P.

Entailment and derivation

Entailment: KB |= Q

- Q is entailed by KB (a set of premises or assumptions) if and only if there is no logically possible world in which Q is false while all the premises in KB are true.
- Or, stated positively, Q is entailed by KB if and only if the conclusion is true in every logically possible world in which all the premises in KB are true.

Derivation: KB |- Q

- We can derive Q from KB if there is a proof consisting of a sequence of valid inference steps starting from the premises in KB and resulting in Q

Two important properties for inference

Soundness: If KB |- Q then KB |= Q

- ❖ If Q is derived from a set of sentences KB using a given set of rules of inference, then Q is entailed by KB.
- ❖ Hence, inference produces only real entailments,
- ❖ or any sentence that follows deductively from the premises is valid.

Completeness: If KB |= Q then KB |- Q

- ❖ If Q is entailed by a set of sentences KB, then Q can be derived from KB using the rules of inference.
- ❖ Hence, inference produces all entailments,
- ❖ or all valid sentences can be proved from the premises.

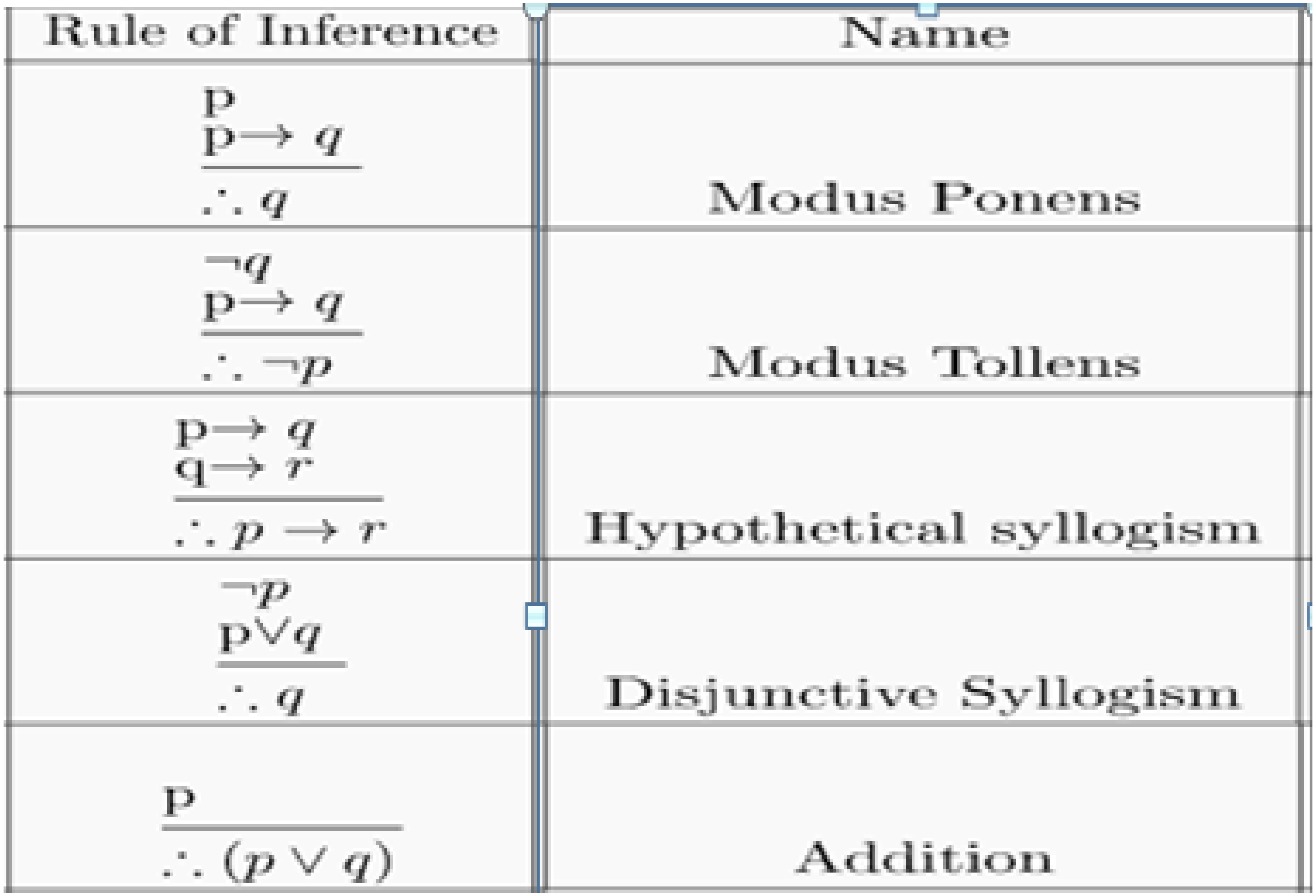
Inference Rule

- Inference rules are the templates for generating valid arguments.
- Inference rules are applied to derive proofs in artificial intelligence, and the proof is a sequence of the conclusion that leads to the desired goal.
- Logical inference is used to create new sentences that logically follow from a given set of predicate calculus sentences (KB).
- An inference algorithm that derives only entailed sentences is called sound or truth-preserving.
- An inference algorithm is complete if it can derive any sentence that is entailed.
- Grounding – the connection between logical reasoning processes and the real environment in which the agent exists.
- Learning – Rules that are produced by a sentence construction process

Inference Rule :: Terminologies

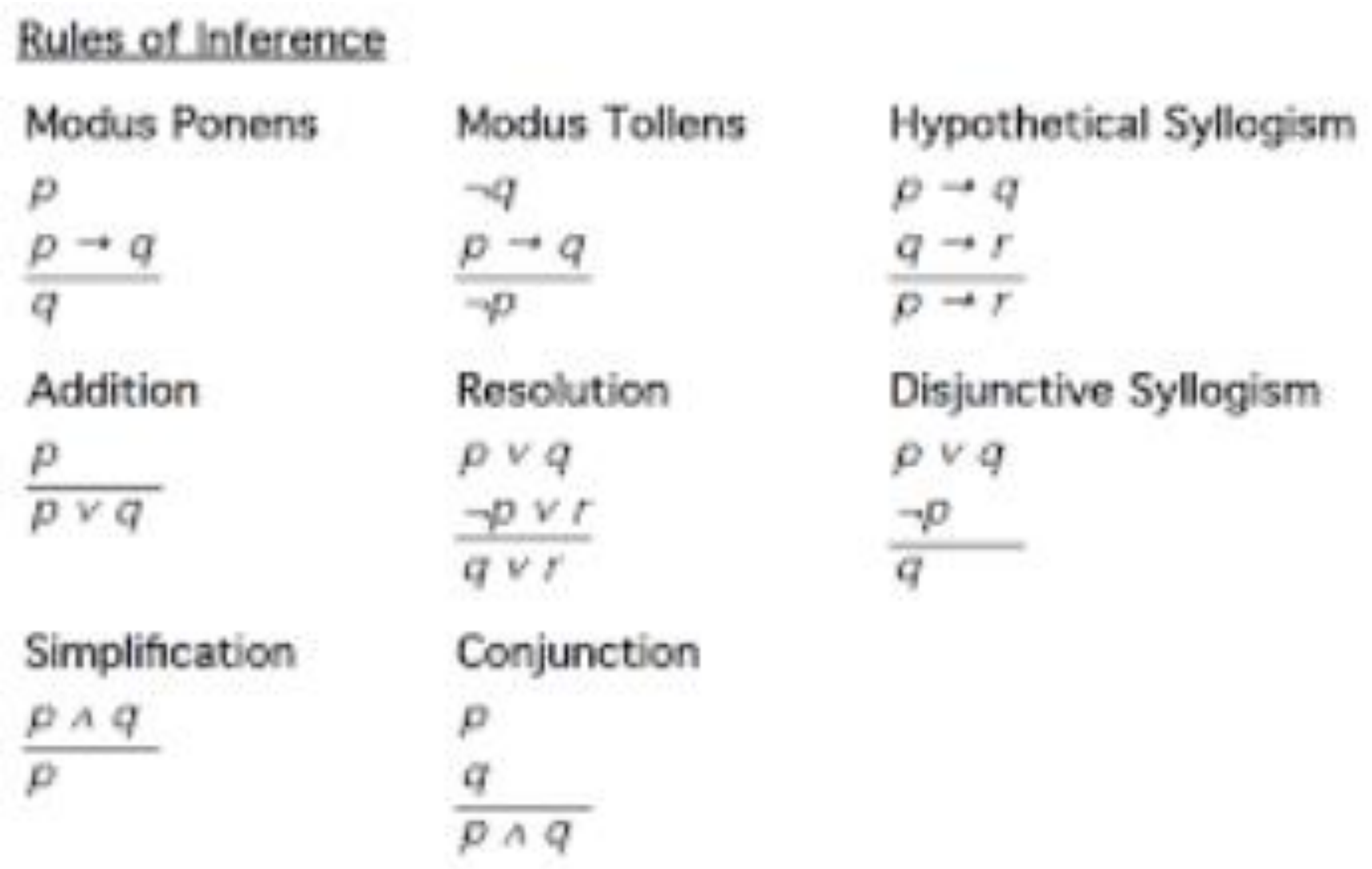
- Implication: It is one of the logical connectives which can be represented as $P \rightarrow Q$. It is a Boolean expression.
- Converse: The converse of implication, which means the right-hand side proposition goes to the left-hand side and vice-versa. It can be written as $Q \rightarrow P$.
- Contrapositive: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

Inference Rule



Rule of Inference	Name
p $p \rightarrow q$ $\therefore q$	Modus Ponens
$\neg q$ $p \rightarrow q$ $\therefore \neg p$	Modus Tollens
$p \rightarrow q$ $q \rightarrow r$ $\therefore p \rightarrow r$	Hypothetical syllogism
$\neg p$ $p \vee q$ $\therefore q$	Disjunctive Syllogism
p $\therefore (p \vee q)$	Addition

Inference Rule



Rules of Inference

Modus Ponens

$$\begin{array}{l} p \\ p \to q \\ \hline q \end{array}$$

Modus Tollens

$$\begin{array}{l} \neg q \\ p \to q \\ \hline \neg p \end{array}$$

Hypothetical Syllogism

$$\begin{array}{l} p \to q \\ q \to r \\ \hline p \to r \end{array}$$

Addition

$$\begin{array}{l} p \\ \hline p \vee q \end{array}$$

Resolution

$$\begin{array}{l} p \vee q \\ \neg p \vee r \\ \hline q \vee r \end{array}$$

Disjunctive Syllogism

$$\begin{array}{l} p \vee q \\ \neg p \\ \hline q \end{array}$$

Simplification

$$\begin{array}{l} p \wedge q \\ \hline p \end{array}$$

Conjunction

$$\begin{array}{l} p \\ q \\ \hline p \wedge q \end{array}$$

Inference Rule

Example. Premises: $\left\{\begin{array}{c}\neg A \rightarrow (C \wedge D) \\ A \rightarrow B \\ \neg B\end{array}\right.$.

Prove: C.

1.	$A \rightarrow B$	Premise
2.	$\neg B$	Premise
3.	$\neg A$	Modus tollens (1,2)
4.	$\neg A \rightarrow (C \wedge D)$	Premise
5.	$C \wedge D$	Modus ponens (3,4)
6.	C	Decomposing a conjunction (5)

Inference Rule

Example. Premises: $\left\{\begin{array}{c} P \wedge Q \\ P \rightarrow \neg(Q \wedge R) \\ S \rightarrow R \end{array}\right.$.

Prove: $\neg S$.

1.	$P \wedge Q$	Premise
2.	P	Decomposing a conjunction (1)
3.	Q	Decomposing a conjunction (1)
4.	$P \rightarrow \neg(Q \wedge R)$	Premise
5.	$\neg(Q \wedge R)$	Modus ponens (3,4)
6.	$\neg Q \vee \neg R$	DeMorgan (5)
7.	$\neg R$	Disjunctive syllogism (3,6)
8.	$S \rightarrow R$	Premise
9.	$\neg S$	Modus tollens (7,8) □

Inference Rule

Example. Premises: $\left\{\begin{array}{c} \neg(A \vee B) \rightarrow C \\ \neg A \\ \neg C \end{array}\right.$.

Prove: B.

1.	$\neg(A \vee B) \rightarrow C$	Premise
2.	$\neg C$	Premise
3.	$A \vee B$	Modus tollens (1,2)
4.	$\neg A$	Premise
5.	B	Disjunctive syllogism (3,4)

Proving things: what are proofs and theorems?

- A proof is a sequence of sentences, where each sentence is either a premise or a sentence derived from earlier sentences in the proof by one of the rules of inference.
- The last sentence is the theorem (also called goal or query) that we want to prove.

Example

(P ^ Q) => R	[“If it is hot and humid, then it is raining”]
Q => P	[“If it is humid, then it is hot”]
Q	[“It is humid.”]

Prove: R

Proving things: what are proofs and theorems?

❖ Example for the “weather problem” given above.

1 Hu	Premise	“It is humid”
2 Hu=>Ho	Premise	“If it is humid, it is hot”
3 Ho	Modus Ponens(1,2)	“It is hot”
4 (Ho^Hu)=>R	Premise	“If it’s hot & humid, it’s raining”
5 Ho^Hu	And Introduction(1,2)	“It is hot and humid”
6 R	Modus Ponens(4,5)	“It is raining”

Proof methods

- ❖ Proof methods divide into (roughly) two kinds:
- ❖ Application of inference rules
  - ➢ Legitimate (sound) generation of new sentences from old
  - ➢ Proof = a sequence of inference rule applications

    Can use inference rules as operators in a standard search algorithm
  - ➢ Typically require transformation of sentences into a normal form
- ❖ Model checking
  - ➢ truth table enumeration (always exponential in n)
  - ➢ improved backtracking, e.g., Davis--Putnam-Logemann-Loveland (DPLL)
  - ➢ heuristic search in model space (sound but incomplete)

    e.g., min-conflicts-like hill-climbing algorithms

Normal forms of PL sentences

- Disjunctive normal form (DNF)
  - Any sentence can be written as a disjunction of conjunctions of literals.
  - Examples: <u>P ^ Q ^ ~R</u>; <u>A^B v C^D v P^Q^R</u>; <u>P</u>
  - Widely used in logical circuit design (simplification)
- Conjunctive normal form (CNF)
  - Any sentence can be written as a conjunction of disjunctions of literals.
  - Examples: <u>P v Q v ~R;</u> <u>(A v B) ^ (C v D) ^ (P v Q v R)</u>; <u>P</u>

Normal forms of PL sentences

- Normal forms can be obtained by applying equivalence laws

[(A v B) => (C v D)] => P

≡ ~[~(A v B) v (C v D)] v P // law for implication

≡ [~~(A v B) ^ ~(C v D)] v P // de Morgan's law

≡ [(A v B)^(~C ^ ~D)] v P // double negation and de Morgan's law

≡ (A v B v P)^(~C^~D v P) // distribution law

≡ (A v B v P)^(~C v P)^(~D v P) // a CNF

Conversion to CNF

- $B_{1,1} \Leftrightarrow (P_{1,2} \vee P_{2,1})$
- Eliminate $\Leftrightarrow$, replacing $\alpha \Leftrightarrow \beta$ with $(\alpha \Rightarrow \beta) \wedge (\beta \Rightarrow \alpha)$.

  $(B_{1,1} \Rightarrow (P_{1,2} \vee P_{2,1})) \wedge ((P_{1,2} \vee P_{2,1}) \Rightarrow B_{1,1})$

- Eliminate $\Rightarrow$, replacing $\alpha \Rightarrow \beta$ with $\neg \alpha \vee \beta$.

  $(\neg B_{1,1} \vee P_{1,2} \vee P_{2,1}) \wedge (\neg(P_{1,2} \vee P_{2,1}) \vee B_{1,1})$

- Move $\neg$ inwards using de Morgan's rules and double-negation:

  $(\neg B_{1,1} \vee P_{1,2} \vee P_{2,1}) \wedge ((\neg P_{1,2} \wedge \neg P_{2,1}) \vee B_{1,1})$

- Apply distributivity law ($\wedge$ over $\vee$) and flatten:

  $(\neg B_{1,1} \vee P_{1,2} \vee P_{2,1}) \wedge (\neg P_{1,2} \vee B_{1,1}) \wedge (\neg P_{2,1} \vee B_{1,1})$

Proof methods

- Definite clause: is a disjunction of literals of which exactly one is positive.
- Horn clause: is a disjunction of literals of which at most one is positive
- Goal clauses: All definite clauses are Horn clauses, as are clauses with no positive literals
- In Horn form
  - The premise is called the body
  - Conclusion is called the head
- A sentence consisting of a single positive literal is called a fact

Horn sentences

- A Horn sentence or Horn clause has the form:

$$P_1 \wedge P_2 \wedge P_3 \ldots \wedge P_n \Rightarrow Q$$

or alternatively

$$\sim P_1 \vee \sim P_2 \vee \sim P_3 \ldots \vee \sim P_n \vee Q$$

where Ps and Q are non-negated atoms

- To get a proof for Horn sentences, apply Modus Ponens repeatedly until nothing can be done
- Can be used with forward chaining or backward chaining.
- These algorithms are very natural and run in linear time

Forward vs. backward chaining

- Forward Chaining: starts with known facts and asserts newly generated fact to reach on to a conclusion (goal).
- Backward Chaining: starts with the goal and works backward to assert what facts (statements) are needed to be asserted to reach on to a conclusion (goal)

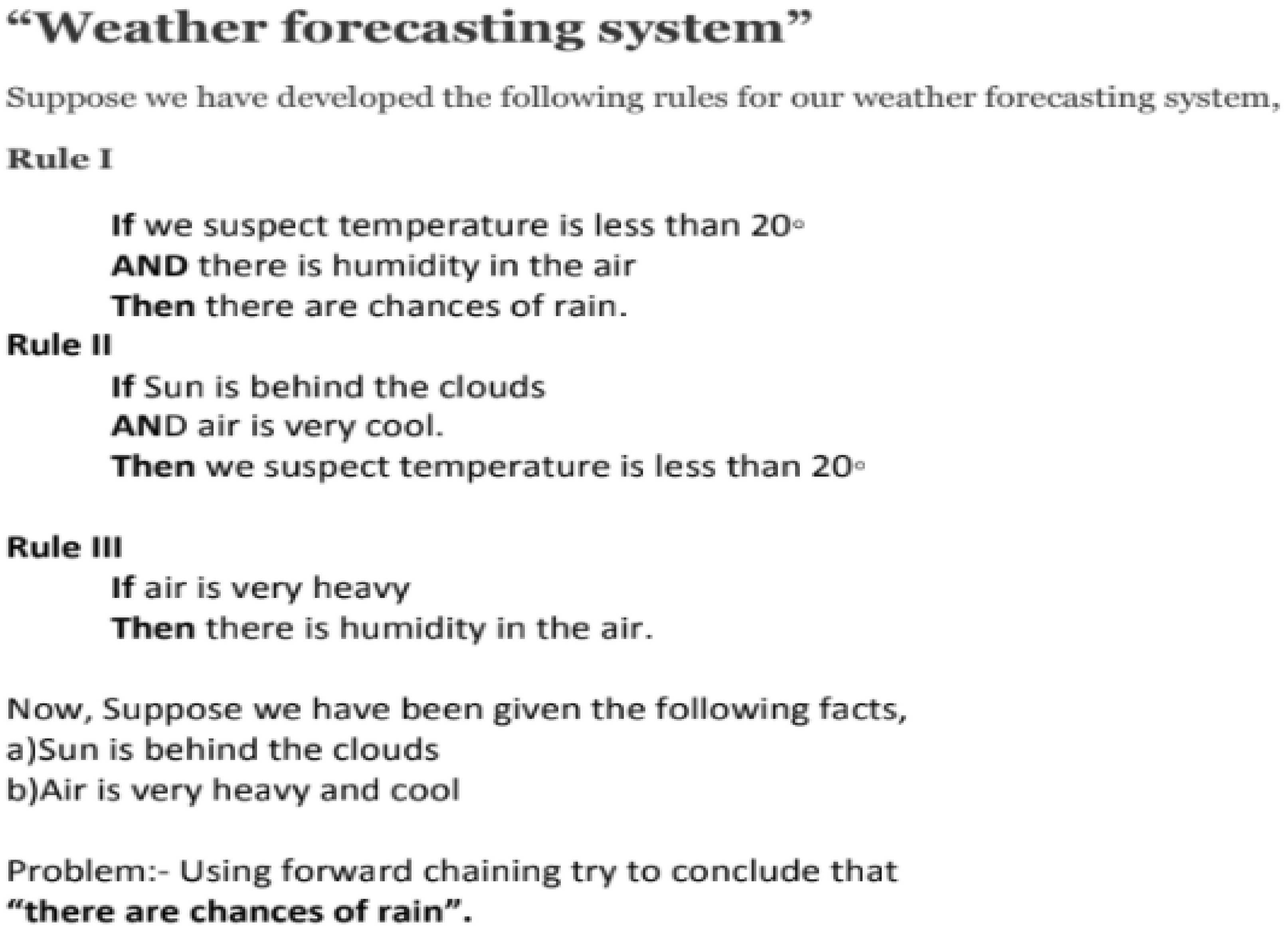
- FC is data-driven, automatic, unconscious processing,

  e.g., object recognition, routine decisions
- May do lots of work that is irrelevant to the goal

- BC is goal-driven, appropriate for problem-solving,

  e.g., Where are my keys? How do I get into a PhD program?
- Complexity of BC can be much less than linear in size of KB

Forward chaining



"Weather forecasting system"

Suppose we have developed the following rules for our weather forecasting system,

Rule I

If we suspect temperature is less than 20°
AND there is humidity in the air
Then there are chances of rain.

Rule II

If Sun is behind the clouds
AND air is very cool.
Then we suspect temperature is less than 20°

Rule III

If air is very heavy
Then there is humidity in the air.

Now, Suppose we have been given the following facts,
a)Sun is behind the clouds
b)Air is very heavy and cool

Problem:- Using forward chaining try to conclude that
"there are chances of rain".

Forward chaining

As we know that forward chaining is a data driven method so we will start from our given data(facts). we can brake our second fact into two facts because it is connecting by AND. So we have three facts-

a) Sun is behind the clouds

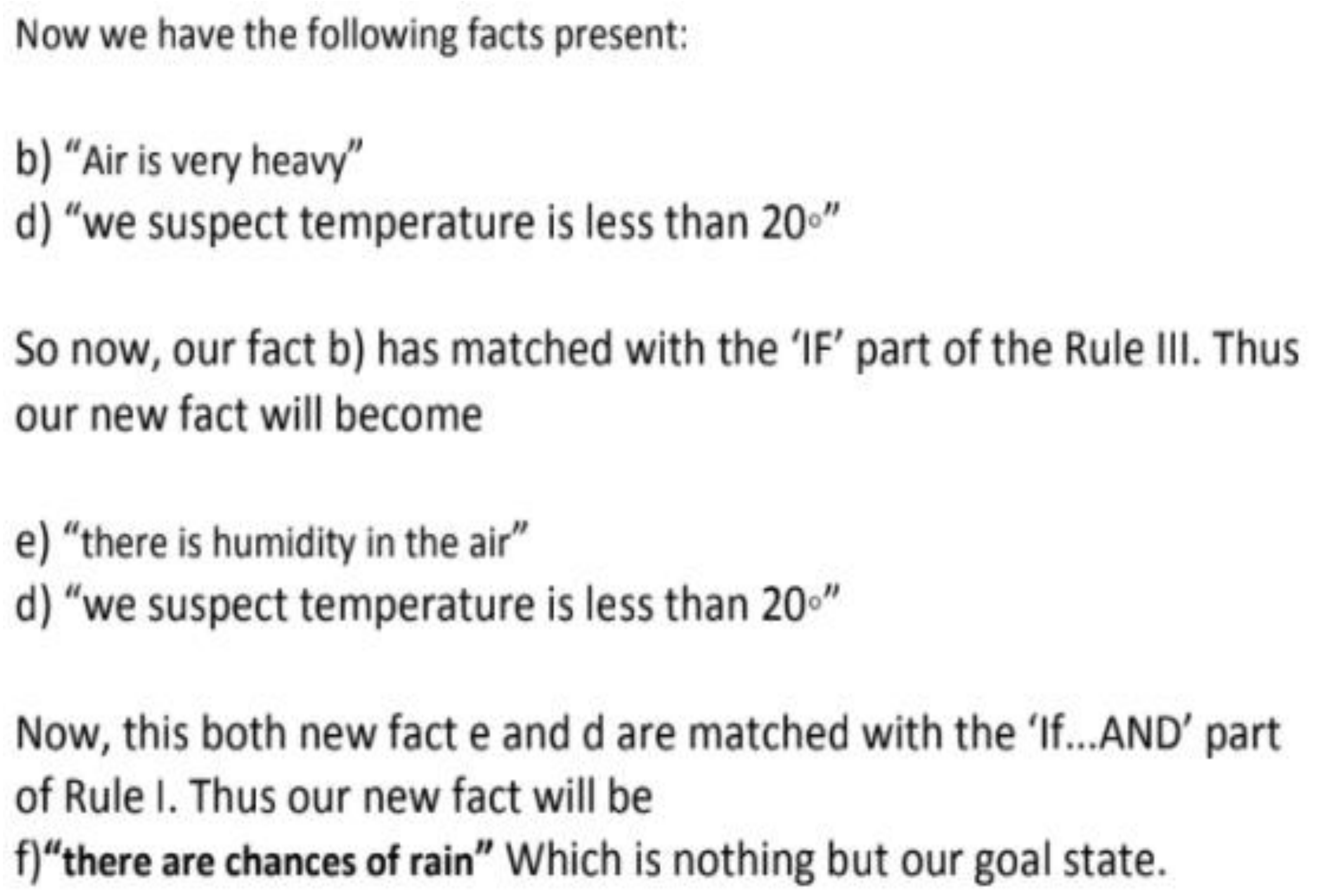
b) Air is very heavy.

c) Air is very cool.

Now, in forward chaining system we will rich goal from the given facts. For this we match with the 'IF..AND' part of the rule base and create new fact which one is present in the 'Then' part. In other words if we ask question **"Why this happened?"** to the given facts then we will get our answer from the set of rules which are present in the knowledge base.

a)Sun is behind the clouds and c) Air is very cool this two facts are present in the IF..AND part of Rule II. Thus we get our new fact "we suspect temperature is less than 20°"

Forward chaining


Now we have the following facts present:

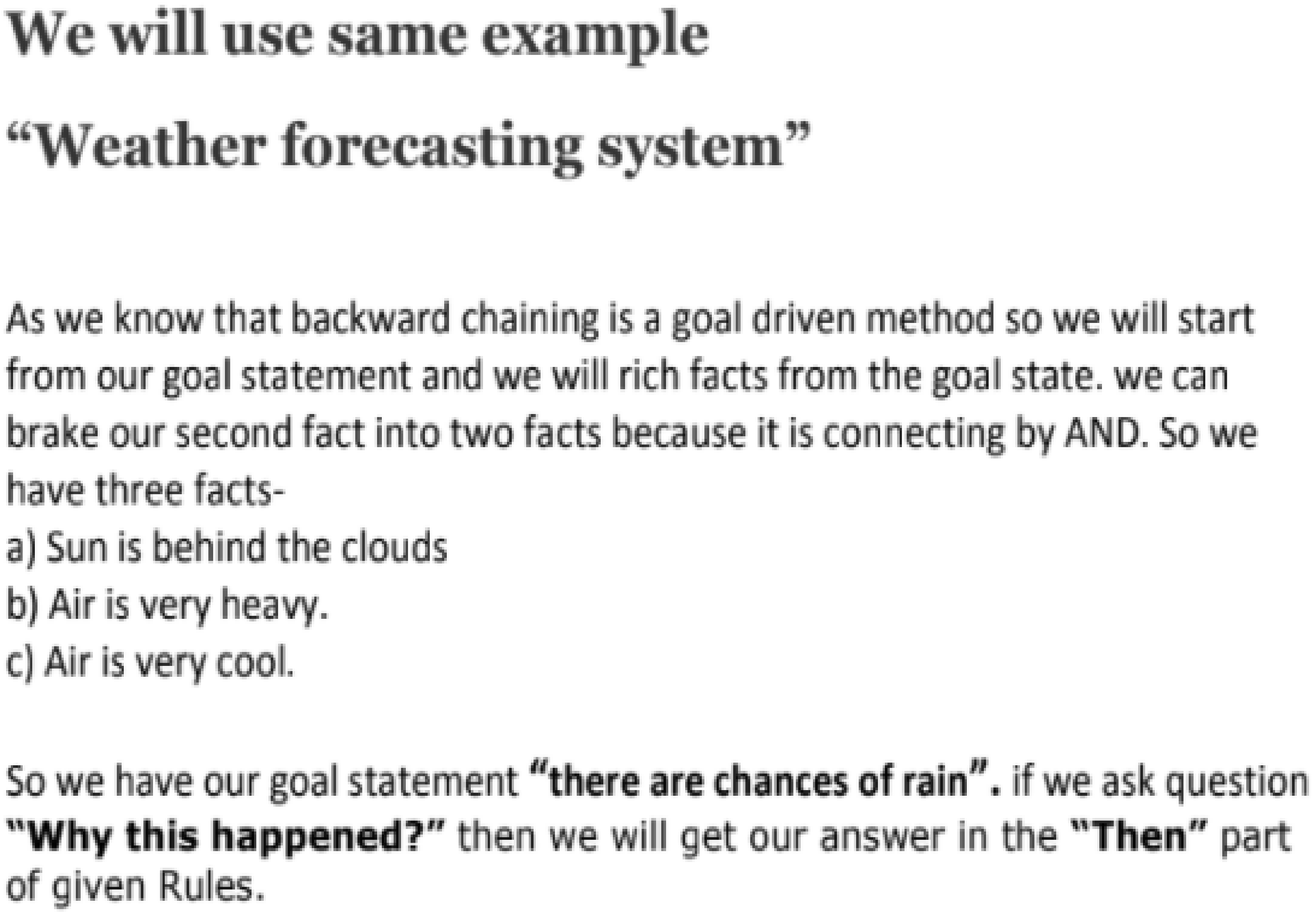
b) "Air is very heavy"
d) "we suspect temperature is less than 20°"

So now, our fact b) has matched with the 'IF' part of the Rule III. Thus our new fact will become

e) "there is humidity in the air"
d) "we suspect temperature is less than 20°"

Now, this both new fact e and d are matched with the 'If...AND' part of Rule I. Thus our new fact will be
f)**"there are chances of rain"** Which is nothing but our goal state.

Backward chaining



We will use same example

“Weather forecasting system”

As we know that backward chaining is a goal driven method so we will start from our goal statement and we will rich facts from the goal state. we can brake our second fact into two facts because it is connecting by AND. So we have three facts-

a) Sun is behind the clouds

b) Air is very heavy.

c) Air is very cool.

So we have our goal statement **“there are chances of rain”.** if we ask question **“Why this happened?”** then we will get our answer in the **“Then”** part of given Rules.

Backward chaining

So the statement "there are chances of rain" is present in the 'Then' part of Rule I. Thus we get our two sub goals

1. "we suspect temperature is less than 20°"
2. "there is humidity in the air"

Now sub goal number 1 is present in the 'Then' part of Rule II. Thus we get our another two sub goals

3."Sun is behind the clouds"

4. "air is very cool". Which is nothing but our first and third fact.(a and c)

And sub goal number 2 is present in the 'Then' part of Rule III. Thus we get our another sub goal

5. "Air is very heavy." Which is nothing but our second fact(b).

Hence we conclude that "**there are chances of rain**" because we rich all facts(data).

Limitation of Propositional Logic

- We cannot represent relations like ALL, some, or none with propositional logic. Example:
  - All the students 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

Propositional logic is a weak language

- Hard to identify "individuals." E.g., Mary, 3
  - Individuals cannot be PL sentences themselves.
- Can't directly talk about properties of individuals or relations between individuals. (hard to connect individuals to class properties).
  - E.g., property of being a human implies property of being mortal
  - E.g. "Bill is tall"
- Generalizations, patterns, regularities can't easily be represented.
  - E.g., all triangles have 3 sides
  - All members of a class have this property
  - Some members of a class have this property
- A better representation is needed to capture the relationship (and distinction) between objects and classes, including properties belonging to classes and individuals.

PREDICATE LOGIC

Predicate Logic

- Logical expressions are built out of components
  - Objects (Constants)
  - Variables
  - Functions
- The above three are called Terms
  - Predicates
  - Connectives
  - Quantifiers

Predicate Logic :: Objects

- Symbols that denote specific things or individuals
  - JOHN
  - MARY
  - BASKETBALL
  - TRIANGLE

Predicate Logic :: Variables

- Unspecific references to objects
  - x
  - y
  - z

Predicate Logic :: Functions

- An argument to a function is either an object or a variable
  - Starting with a lowercase letter
- The value of a function is either an object or a variable
  - exp(0) = 1
  - next-day(THURSDAY) = FRIDAY
  - brother(JOHN) = JIM

Predicate Logic :: Predicates

- Functions which denote attributes of objects or relationships between individuals
    - Starting with a uppercase letter
- Loves(JOHN, MARY)
- Man(SOCRATES)
- Sunny(THURDAY)

Predicate Logic :: Connectives

- Logical operators which computes truth values

  $\wedge$ AND

  $\vee$ OR

  $\neg$ NOT

  $\rightarrow$ IMPLIES

Predicate Logic :: Quantifiers

- Logical operators which assert the scope of a predicate

  ∀ For All (universal quantifier)

  ∃ There Exists (existential quantifier)

- 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).

Predicate Logic :: Basic Elements

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

Predicate Logic

- 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

  Ravi and Ajay are brothers: => Brothers(Ravi, Ajay).

  Chinky is a cat: => Cat (Chinky)

- A well-formed formula (wff) is a sentence containing no “free” variables. That is, all variables are “bound” by universal or existential quantifiers. $(\forall x)P(x,y)$ has x bound as a universally quantified variable, but y is free.

Conversion from Natural Language Sentences to Predicate Logic

- Marcus was a man

  Man(Marcus)

- Marcus was a Pompeian

  Pompeian(Marcus)

- All Pompeians were Romans

  $\forall x\ [\text{Pompeian}(x) \rightarrow \text{Roman}(x)]$

- Caesar was a ruler

  Ruler(Caesar)

- All Romans were either loyal to Caesar or hated him

  $\forall x\ [\text{Roman}(x) \supset (\text{loyalto}(x,\text{Caesar}) \vee \text{hate}(x,\text{Caesar}))]$

- Everyone is loyal to someone

  $\forall x\ \exists y\ \text{loyalto}(x,y)$

Conversion from Natural Language Sentences to Predicate Logic

- People only try to assassinate rulers they aren't loyal to

  $\forall x\ \forall y[(\text{Person}(x) \land \text{Ruler}(y) \land \text{tryassassinate}(x,y)) \rightarrow \neg \text{loyalto}(x,y)]$

- Marcus tried to assassinate Caesar

  tryassassinate(Marcus, Caesar)

Conversion of facts into first-order logic

- John likes all kind of food.
- Apple and vegetable are food
- Anything anyone eats and not killed is food.
- Anil eats peanuts and still alive
- Harry eats everything that Anil eats.
- John likes peanuts.

Conversion of facts into first-order logic

- John likes all kind of food.

  $\forall x: \text{food}(x) \rightarrow \text{likes}(\text{John}, x)$

- Apple and vegetable are food

  $\text{food}(\text{Apple}) \land \text{food}(\text{vegetables})$

- Anything anyone eats and not killed is food.

  $\forall x\ \forall y: [\text{eats}(x, y) \land \neg\ \text{killed}(x)] \rightarrow \text{food}(y)$

- Anil eats peanuts and still alive

  $\text{eats}\ (\text{Anil}, \text{Peanuts}) \land \text{alive}(\text{Anil})$

- Harry eats everything that Anil eats.

  $\forall x: \text{eats}(\text{Anil}, x) \rightarrow \text{eats}(\text{Harry}, x)$

- John likes peanuts.

  likes(John, Peanuts).

Conversion to CNF

- ❖ Standardize variables apart by renaming them: each quantifier should use a different variable.
- ❖ Skolemize: each existential variable is replaced by a Skolem constant or Skolem function of the enclosing universally quantified variables.

 For instance, ∃x Rich(x) becomes Rich(G1) where G1 is a new Skolem constant.

- ❖ Drop universal quantifiers

 For instance, ∀ x Person(x) becomes Person(x).

Convert FOL statements into CNF

Eliminate all implication (→) and rewrite

- ∀x: ¬ food(x) V likes(John, x)
- food(Apple) Λ food(vegetables)
- ∀x ∀y: ¬ [eats(x, y) Λ ¬ killed(x)] V food(y)
- eats (Anil, Peanuts) Λ alive(Anil)
- ∀x: ¬ eats(Anil, x) V eats(Harry, x)
- likes(John, Peanuts).

Convert FOL statements into CNF

Move negation (¬)inwards and rewrite

- ∀x: ¬ food(x) V likes(John, x)
- food(Apple) Λ food(vegetables)
- ∀x ∀y: ¬ eats(x, y) V killed(x) V food(y)
- eats (Anil, Peanuts) Λ alive(Anil)
- ∀x: ¬ eats(Anil, x) V eats(Harry, x)
- likes(John, Peanuts).

Convert FOL statements into CNF

Rename variables or standardize variables

- ∀x: ¬ food(x) V likes(John, x)
- food(Apple) Λ food(vegetables)
- ∀y ∀z: ¬ eats(y, z) V killed(y) V food(z)
- eats (Anil, Peanuts) Λ alive(Anil)
- ∀w: ¬ eats(Anil, w) V eats(Harry, w)
- likes(John, Peanuts).

Convert FOL statements into CNF

Eliminate existential instantiation quantifier by elimination.

- ❖ In this step, we will eliminate existential quantifier ∃, and this process is known as Skolemization.
- ❖ But in this example problem since there is no existential quantifier so all the statements will remain same in this step.

Convert FOL statements into CNF

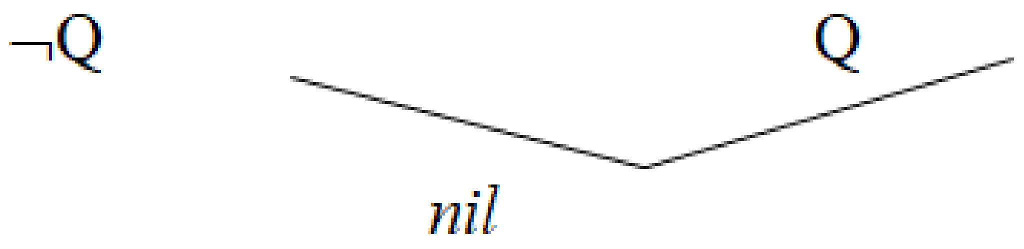
Drop Universal quantifiers.

In this step we will drop all universal quantifier since all the statements are not implicitly quantified so we don't need it.

- ¬ food(x) V likes(John, x)
- food(Apple)
- food(vegetables)
- ¬ eats(y, z) V killed(y) V food(z)
- eats (Anil, Peanuts)
- alive(Anil)
- ¬ eats(Anil, w) V eats(Harry, w)
- likes(John, Peanuts).

Resolution Proof – Generating *nil*

Consider:



- Resolution is a way of finding contradictions in a database of clauses with minimum use of substitution. Resolution is a proof technique that works on conjunctive normal form expressions. It involves:
    - Selecting two clauses that contain conflicting terms.
    - Combining the terms contained in those two clauses.
    - Cancelling the terms that conflict
- It is used for proving the satisfiability of a sentence.

Resolution Proof – Generating *nil*

```
resolve-Boolean(A, ST) =
1. Construct L, the list of of clauses from A.
2. Negate ST, convert the result to conjunctive normal form,
   and add the resulting clauses to L.
3. Until either nil is generated or no progress is being made
   do:
        3. 1 Choose two parent clauses.
        3. 2 Resolve the parent clauses together.
        3. 3 If the resolvent is not nil and is not in L, add
             it to L.
4. If nil was generated, a contradiction has been found. Return
   success. ST must be true.
5. If nil was not generated and there was nothing left to do,
   return failure.
```

Resolution Proof – Example

Prove R given:

Given Axioms:	***Clauses:***
P	P
$(P \wedge Q) \rightarrow R$	$\neg P \vee \neg Q \vee R$
$(S \vee T) \rightarrow Q$	$\neg S \vee Q$
	$\neg T \vee Q$
T	T

Add:

$\neg R$	$\neg R$

Resolution Proof – Example

P

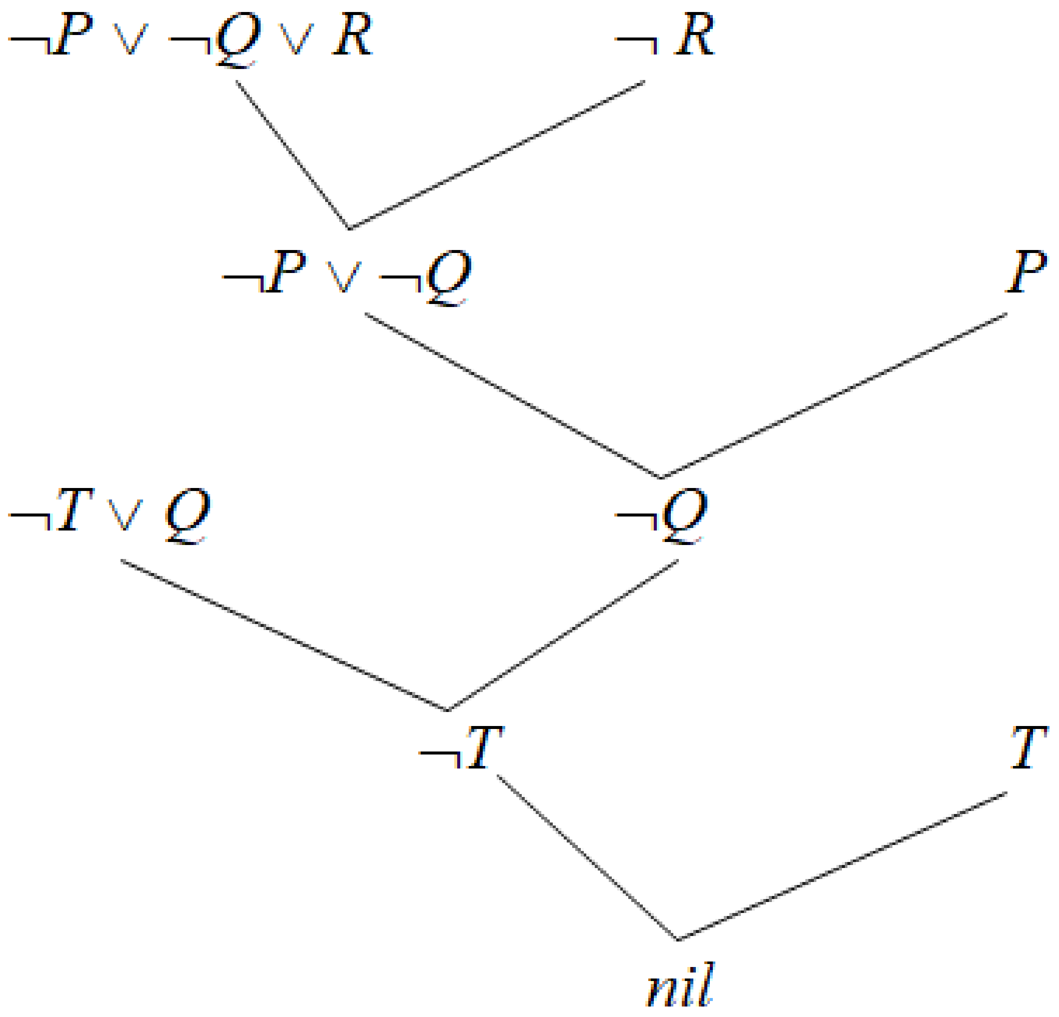
$\neg P \vee \neg Q \vee R$

$\neg S \vee Q$

$\neg T \vee Q$

T

$\neg R$



Resolution Proof

Marcus was a man.

Marcus was a Pompeian.

All Pompeians were Romans.

Caesar was a ruler.

All Romans were either loyal to Caesar or hated him.

Everyone is loyal to someone.

People only try to assassinate rulers they aren't loyal to.

Marcus tried to assassinate Caesar.

Prove that Marcus hated Caesar.

Resolution Proof

1. Marcus was a man.

   man(Marcus)

2. Marcus was a Pompeian

   pompeian(Marcus)

3. All Pompeians were Romans.

   $\forall x$: pompeian(x) $\rightarrow$ roman(x)

   ~pompeian(x) v roman(x)

4. Caesar was a ruler.

   ruler(Caesar)

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

   $\forall x$: roman(x) $\rightarrow$ loyalto(x, Caesar) v hate(x, Caesar)

   ~roman(y) v loyalto(y, Caesar) v hate(y, Caesar)

Resolution Proof

6. Everyone is loyal to someone.

   $\forall x\ \exists y$: loyalto(x, y)

   loyalto(z, f(z))   // f is a skolem function and returns a person that z is loyal to

7. People only try to assassinate rulers they aren't loyal to.

   $\forall x\ \forall y$: (man(a) $\wedge$ ruler(b) $\wedge$ tryassassinate(a,b)) $\rightarrow$ $\neg$loyalto(a,b)]

   $\neg$(man(a) $\wedge$ ruler(b) $\wedge$ tryassassinate(a,b)) v $\neg$ loyalto(a, b)

   = $\neg$ man(a) v $\neg$ ruler(b) v $\neg$ trytoassassinate(a, b) v $\neg$ loyalto(a, b)

8. Marcus tried to assassinate Caesar.

   trytoassassinate(Marcus, Caesar)

Prove hated(Marcus, Caesar).

Resolution Proof

- Introduce 9. ~hated(Marcus, Caesar).

  Resolve 9 with 5 unifying Marcus to y yields

- ~roman(Marcus) v loyalto(Marcus, Caesar)

  Resolve 10 with 3 unifying Marcus to x yields

- ~pompeian(Marcus) v loyalto(Marcus, Caesar)

  Resolve 11 with 2 yields

- loyalto(Marcus, Caesar)

  Resolve 12 with 7 unifying Marcus to a and Caesar to b yields

- ~man(Marcus) v ~ruler(Caesar) v ~trytoassassinate(Marcus,Caesar)

  Resolve 13 with 1 yields

- ~ruler(Caesar) v ~trytoassassinate(Marcus, Caesar)

  Resolve 14 with 4 yields

- ~ trytoassassinate(Marcus, Caesar)

Resolution Proof

Resolve 15 with 8 yields the null hypothesis.

Therefore, ~hate(Marcus, Caesar) is false, so hate(Marcus, Caesar) is true.

Resolution Proof - Practice

Marcus was a man.

Marcus was a Pompeian.

All Pompeians were Romans.

Caesar was a ruler.

All Romans were either loyal to Caesar or hated him.

Everyone is loyal to someone.

People only try to assassinate rulers they aren't loyal to.

Marcus tried to assassinate Caesar.

Was Marcus loyal to Caesar?

Resolution Proof - Practice

(1) *Marcus was a man.*

$\text{man}(\text{Marcus})$

(2) *Marcus was a Pompeian.*

$\text{Pompeian}(\text{Marcus})$

(3) *Marcus was born in 40 A.D.*

$\text{born}(\text{Marcus}, 40)$

(4) *All men are mortal.*

$\forall x\ \text{man}(x) \rightarrow \text{mortal}(x)$

(5, 6) *AllPompeians died when the volcano erupted in 79 A.D.*

$\text{erupted}(\text{volcano}, 79) \wedge \forall x\ \text{Pompeian}(x) \rightarrow \text{died}(x, 79)$

(7) *No mortal lives longer than 150 years.*

$\forall x\ \forall t_1\ \forall t_2\ \text{mortal}(x) \wedge \text{born}(x, t_1) \wedge \text{gt}(t_2 - t_1, 150) \rightarrow \text{dead}(x, t_2)$

(8) *It is now 2004.*

$\text{now} = 2004$

Is Marcus alive?

Resolution Proof - Practice

Given:

$\forall x\ [Roman(x) \wedge know(x, Marcus)] \rightarrow$

$\quad [hate(x, Caesar) \vee (\forall y\ (\exists z\ hate(y, z)) \rightarrow thinkcrazy(x, y))]$

$Roman(Isaac)$

$\neg hate(Isaac, Caesar)$

$hate(Paulus, Marcus)$

$\neg thinkcrazy(Isaac, Paulus)$

Prove:

$\neg know(Isaac, Marcus)$

Quantifier Scope

- Switching the order of universal quantifiers *does not* change the meaning:

  $(\forall x)(\forall y)P(x,y) \leftrightarrow (\forall y)(\forall x)\ P(x,y)$
- Similarly, you can switch the order of existential quantifiers:

  $(\exists x)(\exists y)P(x,y) \leftrightarrow (\exists y)(\exists x)\ P(x,y)$
- Switching the order of universals and existentials *does* change meaning:
  - Everyone likes someone: $(\forall x)(\exists y)$ likes(x,y)
  - Someone is liked by everyone: $(\exists y)(\forall x)$ likes(x,y)

Connections between All and Exists

- We can relate sentences involving $\forall$ and $\exists$ using De Morgan’s laws:

$$(\forall x)\ \neg P(x) \leftrightarrow \neg(\exists x)\ P(x)$$

$$\neg(\forall x)\ P(x) \leftrightarrow (\exists x)\ \neg P(x)$$

$$(\forall x)\ P(x) \leftrightarrow \neg\ (\exists x)\ \neg P(x)$$

$$(\exists x)\ P(x) \leftrightarrow \neg(\forall x)\ \neg P(x)$$

Disclaimer

The material for the presentation has been compiled from various sources such as prescribed text books by Russell and Norvig and other tutorials and lecture notes. The information contained in this lecture/ presentation is for educational purpose only.

Thank You *for* Your Attention !