4

Logic Concepts and Logic Programming

4.1 Introduction

Initially, logic was considered to be a branch of philosophy; however, since the middle of t nineteenth century, *formal logic* has been studied in the context of foundations of mathemati where it was often referred to as *symbolic logic*. Logic helps in investigating and classifying t structure of statements and arguments through the study of formal systems of inference. It therefore a study of methods that help in distinguishing correct reasoning from incorrect reaso ing. Formally, logic is concerned with the principles of drawing valid inferences from a given s of true statements. The development of formal logic and its implementation in computing machi ery is fundamental to the study and growth of computer science. Formal logic deals with the stu of inference with purely formal content; it is often used as a synonym for symbolic logic, which the study of symbolic abstractions. Symbolic logic is often divided into two branches, namel *propositional logic* and *predicate logic*. In the study of logic, *a proposition* refers to a declarat statement that is either true or false (but not both) in a given context. One can infer a new prop sition from a given set of propositions in the same context using logic. An extension to symbo logic is *mathematical logic*, which is particularly concerned with the study of proof theory, s theory, model theory, and recursion theory. The field of logic is also concerned with core top such as the study of validity, consistency, and inconsistency. Logical systems should poss properties such as consistency, soundness, and completeness. Consistency implies that none the theorems of the system should contradict each other; soundness means that the inference ru shall never allow a false inference from true premises. If a system is sound and its axioms are t then its theorems are also guaranteed to be true. Completeness means that there are no t sentences in the system that cannot be proved in the system.

In this chapter, the concepts of propositional calculus and logic are introduced along with four formal methods concerned with proofs and deductions. The concept of propositional logic has also been extended to first-order predicate logic followed by the evolution of logic programming which forms the basis of the logic programming language called PROLOG (this language has been described in detail in the next chapter) (Kaushik S., 2002).

4.2 Propositional Calculus

Propositional calculus (PC) refers to a language of propositions in which a set of rules are used to combine simple propositions to form compound propositions with the help of certain logical operators. These logical operators are often called *connectives*; examples of some connectives are *not* (~), *and* (Λ), *or* (V), *implies* (→), and *equivalence* (↔). In PC, it is extremely important to understand the concept of a well-formed formula. A well-formed formula is defined as a symbol or a string of symbols generated by the formal grammar of a formal language. The following are some important properties of a well-formed formula in PC:

- The smallest unit (or an atom) is considered to be a well-formed formula.
- If α is a well-formed formula, then $\sim\alpha$ is also a well-formed formula.
- If α and β are well-formed formulae, then $(\alpha \wedge \beta)$, $(\alpha \vee \beta)$, $(\alpha \rightarrow \beta)$, and $(\alpha \leftrightarrow \beta)$ are also well-formed formulae.

A propositional expression is called a well-formed formula if and only if it satisfies the above properties.

4.2.1 Truth Table

In PC, a *truth table* is used to provide operational definitions of important logical operators; it elaborates all possible truth values of a formula. The logical constants in PC are *true* and *false* and these are represented as T and F, respectively in a truth table. Let us assume that *A*, *B*, *C*, ... are propositioned symbols. The meanings of above-mentioned logical operators are given in a truth table (Table 4.1) as follows:

Table 4.1 Truth Table for Logical Operators

A	B	~A	A Λ B	A V B	A → B	A ↔ B
T	T	F	T	T	T	T
T	F	F	F	T	F	F
F	T	T	F	T	T	F
F	F	T	F	F	T	T

The truth values of well-formed formulae are calculated by using the truth table approach. Let consider the following example.

Example 4.1 Compute the truth value of $\alpha : (A \vee B) \wedge (\sim B \rightarrow A)$ using truth table approach.

Solution Using the truth table approach, let us compute truth values of $(A \vee B)$ and $(\sim B \rightarrow A)$ and th compute for the final expression $(A \vee B) \wedge (\sim B \rightarrow A)$ (as given in Table 4.2).

Table 4.2 Truth Table for α

A	B	A V B	~B	~B → A	α
T	T	T	F	T	**T**
T	F	T	T	T	**T**
F	T	T	F	T	**T**
F	F	F	T	F	**F**

Definition: Two formulae α and β are said to be logically equivalent $(\alpha \equiv \beta)$ if and only if t truth values of both are the same for all possible assignments of logical constants (T or F) to t symbols appearing in the formulae.

4.2.2 Equivalence Laws

Equivalence relations (or laws) are used to reduce or simplify a given well-formed formula or derive a new formula from the existing formula. Some of the important equivalence laws a given in Table 4.3. These laws can be verified using the truth table approach.

Table 4.3 Equivalence Laws

Name of Relation	Equivalence Relations
Commutative Law	$A \vee B \equiv B \vee A$ $A \wedge B \equiv B \wedge A$
Associative Law	$A \vee (B \vee C) \equiv (A \vee B) \vee C$ $A \wedge (B \wedge C) \equiv (A \wedge B) \wedge C$
Double Negation	$\sim(\sim A) \equiv A$
Distributive Laws	$A \vee (B \wedge C) \equiv (A \vee B) \wedge (A \vee C)$ $A \wedge (B \vee C) \equiv (A \wedge B) \vee (A \wedge C)$
De Morgan's Laws	$\sim(A \vee B) \equiv \sim A \wedge \sim B$ $\sim(A \wedge B) \equiv \sim A \vee \sim B$

(Co

Table 4.3 (Contd.)

Name of Relation	**Equivalence Relations**
Absorption Laws	$A \vee (A \wedge B) \cong A$ $A \wedge (A \vee B) \cong A$ $A \vee (\sim A \wedge B) \cong A \vee B$ $A \wedge (\sim A \vee B) \cong A \wedge B$
Idempotence	$A \vee A \cong A$ $A \wedge A \cong A$
Excluded Middle Law	$A \vee \sim A \cong$ T (True)
Contradiction Law	$A \wedge \sim A \cong F$ (False)
Commonly used equivalence relations	$A \vee F \cong A$ $A \vee T \cong T$ $A \wedge T \cong A$ $A \wedge F \cong F$ $A \rightarrow B \cong \sim A \vee B$ $A \leftrightarrow B \cong (A \rightarrow B) \wedge (B \rightarrow A)$ $\cong (A \wedge B) \vee (\sim A \wedge \sim B)$

Let us verify the absorption law $A \vee (A \wedge B) \cong A$ using truth table approach as shown in Table 4.4.

Table 4.4 Verification of $A \vee (A \wedge B) \cong A$

A	**B**	**A Λ B**	**A V A Λ B**
T	T	T	**T**
T	F	F	**T**
F	T	F	**F**
F	F	F	**F**

We can clearly see that the truth values of $A \vee (A \wedge B)$ and A are same; therefore, these expressions are equivalent.

4.3 Propositional Logic

Propositional logic (*or prop logic*) deals with the validity, satisfiability (also called consistency), and unsatisfiability (inconsistency) of a formula and the derivation of a new formula using equivalence laws. Each row of a truth table for a given formula α is called its *interpretation* under which the value of a formula may be either *true* or *false*. A formula α is said to be a *tautology* if and only if the value of α is true for all its interpretations. Now, the validity, satisfiability, and unsatisfiability of a formula may be determined on the basis of the following conditions:

- A formula α is said to be *valid* if and only if it is a *tautology*.
- A formula α is said to be *satisfiable* if there exists at least one interpretation for which α is true.
- A formula α is said to be *unsatisfiable* if the value of α is false under all interpretations.

Let us consider the following example to explain the concept of validity:

Example 4.2 Show that the following is a valid argument:
If it is humid then it will rain and since it is humid today it will rain

Solution Let us symbolize each part of the above English sentence by propositional a
as follows:

A : It is humid
B : It will rain

Now, the formula (α) corresponding to the given sentence:

If it is humid then it will rain and since it is humid today it will rain

may be written as

$$\alpha : [(A \to B) \wedge A] \to B$$

Using the truth table approach (as given in Table 4.5), one can see that α is true under all interpretations
hence is a *valid argument*.

Table 4.5 Truth Table for $[(A \to B) \wedge A] \to B$

A	B	A → B = (X)	X ∧ A = (Y)	Y → B
T	T	T	T	T
T	F	F	F	T
F	T	T	F	T
F	F	T	F	T

The truth table approach is a simple and straightforward method and is extremely usef
presenting an overview of all the *truth* values in a given situation. Although it is an easy m
for evaluating consistency, inconsistency, or validity of a formula, the limitation of this m
lies in the fact that the size of truth table grows exponentially. That is, if a formula conta
atoms, then the truth table will contain 2^n entries. Moreover, it may be possible in some cases
all entries of a truth table are not required. In such situations, the construction of a truth t
becomes a futile exercise.

For example, if we have to show that a formula $\alpha : (A \wedge B \wedge C \wedge D) \to (B \vee E)$ is *valid* usin
truth table approach, then we need to construct a table containing 32 rows and compute the
values of α for all 32 interpretations. In this example, we notice that the value of $(A \wedge B \wedge C$
is false for 30 out of the 32 entries and is true for 2 entries only. Since we know that $(X \to$

true in all cases except the one in which $X = T$ and $Y = F$, it is clear that α is true for these 30 cases where $(A \land B \land C \land D)$ is *false*. Hence, we are left to verify whether α is true or not for 2 entries only by checking an expression on the right side of $\rightarrow$. If this expression is true then α is valid, otherwise it is not valid.

Use of the truth table approach in such situations proves to be a wastage of time. Therefore, we require some other methods which can help in proving the validity of the formula directly. Some other methods that are concerned with proofs and deductions are as follows:

- Natural deduction system
- Axiomatic system
- Semantic tableau method
- Resolution refutation method

All these methods have been discussed in the following sections.

4.4 Natural Deduction System

Natural deduction system (NDS) is thus called because of the fact that it mimics the pattern of natural reasoning. This system is based on a set of deductive inference rules. Assuming that $A_1, \ldots, A_k$, where $1 \le k \le n$, are a set of atoms and α_j, where $1 \le j \le m$, and β are well-formed formulae, the inference rules may be stated as shown in the following NDS rules table (Table 4.6).

Table 4.6 NDS Rules Table

Rule Name	**Symbol**	**Rule**	**Description**
Introducing $\land$	$(I{:}\land)$	If $A_1, \ldots, A_n$ then $A_1 \land \ldots \land A_n$	If $A_1, \ldots, A_n$ are true, then their conjunction $A_1 \land \ldots \land A_n$ is also *true*.
Eliminating $\land$	$(E{:}\land)$	If $A_1 \land \ldots \land A_n$ then A_i $(1 \le i \le n)$	If $A_1 \land \ldots \land A_n$ is *true*, then any A_i is also *true*.
Introducing V	$(I{:}\text{V})$	If any A_i $(1 \le i \le n)$ then $A_1 \text{ V} \ldots \text{ V } A_n$	If any A_i $(1 \le i \le n)$ is *true*, then $A_1 \text{ V} \ldots \text{ V } A_n$ is also *true*.
Eliminating V	$(E{:}\text{V})$	If $A_1 \text{ V} \ldots \text{ V } A_n, A_1 \rightarrow A, \ldots, A_n \rightarrow A$ then A	If $A_1 \text{ V} \ldots \text{ V } A_n, A_1 \rightarrow A, A_2 \rightarrow A, \ldots,$ and $A_n \rightarrow A$ are *true*, then A is *true*.
Introducing $\rightarrow$	$(I : \rightarrow)$	If from $\alpha_1, \ldots, \alpha_n$ infer β is proved then $\alpha_1 \land \ldots \land \alpha_n \rightarrow \beta$ is proved	If given that $\alpha_1, \alpha_2, \ldots,$ and α_n are *true* and from these we deduce β then $\alpha_1 \land \ldots \land \alpha_n \rightarrow \beta$ is also *true*.

(Contd.)

Table 4.6 (Contd.)

Rule Name	Symbol	Rule	Description
Eliminating $\rightarrow$	$(E: \rightarrow)$	If $A_1 \rightarrow A, A_1$, then A	If $A_1 \rightarrow A$ and A_1 are *true* then A is also *tru* This is called *Modus Ponen* rule.
Introducing $\leftrightarrow$	$(I: \leftrightarrow)$	If $A_1 \rightarrow A_2$, $A_2 \rightarrow A_1$ then $A_1 \leftrightarrow A_2$	If $A_1 \rightarrow A_2$ and $A_2 \rightarrow A_1$ are *true* then $A_1 \leftrightarrow$ A_2 is also *true*.
Elimination $\leftrightarrow$	$(E: \leftrightarrow)$	If $A_1 \leftrightarrow A_2$ then $A_1 \rightarrow A_2, A_2 \rightarrow A_1$	If $A_1 \leftrightarrow A_2$ is *true* then $A_1 \rightarrow A_2$ and $A_2 \rightarrow$ A_1 are *true*
Introducing ~	$(I: \sim)$	*If from A infer* $A_1 \wedge \sim A_1$ is proved then $\sim A$ is proved	If from A (which is *true*), a contradiction is proved then truth of $\sim A$ is also proved
Eliminating ~	$(E: \sim)$	*If from* $\sim A$ *infer* $A_1 \wedge \sim A_1$ is proved then A is proved	If from $\sim A$, a contradiction is proved then truth of A is also proved

A theorem in the NDS written as *from* $\alpha_1, \ldots, \alpha_n$ *infer* β leads to the interpretation that β deduced from a set of hypotheses $\{\alpha_1, \ldots, \alpha_n\}$. All hypotheses are assumed to be true in a giv context and therefore the theorem β is also true in the same context. Thus, we can conclude tha is consistent. A theorem that is written as *infer* β implies that there are no hypotheses and β is tr under all interpretations, i.e., β is a *tautology* or *valid*. Let us consider the following example a show the proof using Natural deduction systems. The conventions used in such a proof are follows:

- The 'Description' column consists of rules applied on a subexpression in the proof line.
- The second column consists the subexpression obtained after applying an appropriate rule.
- The final column consists the line number of subexpressions in the proof.

xample 4.3 Prove that $A \wedge (B \vee C)$ is deduced from $A \wedge B$.

olution The theorem in NDS can be written as *from* $A \wedge B$ *infer* $A \wedge (B \vee C)$ in NDS. We can prove heorem (Table 4.7) as follows:

Table 4.7 Proof of the Theorem for Example 4.3

Description	Formula	Comments
Theorem	*from* $A \wedge B$ *infer* $A \wedge (B \vee C)$	*To be proved*
Hypothesis (given)	$A \wedge B$	1
$E: \wedge$ (1)	A	2
$E: \wedge$ (1)	B	3
$I: \vee$ (3)	$B \vee C$	4
$I: \wedge$ (2, 4)	$A \wedge (B \vee C)$	Proved

If we assume that $\alpha \rightarrow \beta$ is *true*, then we can conclude that β is also *true* if α is *true*. It can be represented in the form of a theorem of NDS as *from* α *infer* β and if we can prove the theorem then we can conclude the truth of $\alpha \rightarrow \beta$. The converse of this is also true. Let us state formally the deduction theorem in NDS.

Deduction Theorem To prove a formula $\alpha_1 \Lambda \ldots \Lambda \alpha_n \rightarrow \beta$, it is sufficient to prove a theorem *from* $\alpha_1, \ldots, \alpha_n$ *infer* β. Conversely, if $\alpha_1 \Lambda \ldots \Lambda \alpha_n \rightarrow \beta$, is proved then the theorem *from* $\alpha_1, \ldots, \alpha_n$, *infer* β is assumed to be proved.

Let us consider the following example to show the use of deduction theorem.

Example 4.4 Prove the theorem *infer* $[(A \rightarrow B) \Lambda (B \rightarrow C)] \rightarrow (A \rightarrow C)$.

Solution The theorem *infer* $[(A \rightarrow B) \Lambda (B \rightarrow C)] \rightarrow (A \rightarrow C)$ is reduced to the theorem *from* $(A \rightarrow B), (B \rightarrow C)$ *infer* $(A \rightarrow C)$ using deduction theorem. Further, to prove '$A \rightarrow C$', we will have to prove a sub-theorem *from* A *infer* C. The proof of the theorem is shown in Table 4.8.

Table 4.8 Proof of the Theorem *from* $(A \rightarrow B), (B \rightarrow C)$ *infer* $(A \rightarrow C)$

Description	**Formula**	**Comments**	
Theorem	*from* $A \rightarrow B, B \rightarrow C$ *infer* $A \rightarrow C$	To be proved	
Hypothesis 1	$A \rightarrow B$	1	
Hypothesis 2	$B \rightarrow C$	2	
Sub-theorem	*from* A *infer* C	3	
Hypothesis	A		3.1
E: $\rightarrow$ (1, 3.1)	B		3.2
E: $\rightarrow$ (2, 3.2)	C		3.3
I: $\rightarrow$ (3)	$A \rightarrow C$	Proved	

4.5 Axiomatic System

The *axiomatic system* is based on a set of three axioms and one rule of deduction. Although minimal in structure, it is as powerful as the truth table and NDS approaches. In axiomatic system, the proofs of the theorems are often difficult and require a guess in selection of appropriate axiom(s). In this system, only two logical operators *not* (~) and *implies* ($\rightarrow$) are allowed to form a formula. It should be noted that other logical operators, such as Λ; V, and $\leftrightarrow$, can be easily expressed in terms of ~ and $\rightarrow$ using equivalence laws stated earlier. For example,

$A \Lambda B \cong \sim(\sim A \vee \sim B) \cong \sim(A \rightarrow \sim B)$

$A \vee B \cong \sim A \rightarrow B$

$A \leftrightarrow B \cong (A \rightarrow B) \Lambda (B \rightarrow A) \cong \sim[(A \rightarrow B) \rightarrow \sim(B \rightarrow A)]$

In axiomatic system, there are three axioms, which are always true (or valid), and one rule cal *modus ponen* (MP). Here, α, β, and γ are well-formed formulae of the axiomatic system. T three axioms and the rule are stated as follows:

Axiom 1 $\alpha \to (\beta \to \alpha)$
Axiom 2 $[\alpha \to (\beta \to \gamma)] \to [(\alpha \to \beta) \to (\alpha \to \gamma)]$
Axiom 3 $(\sim\alpha \to \sim\beta) \to (\beta \to \alpha)$

Modus Ponen Rule *Hypotheses:* $\alpha \to \beta$ and α; *Consequent:* β

Interpretation of Modus Ponen Rule: Given that $\alpha \to \beta$ and α are hypotheses (assumed to *true*), β is inferred (i.e., *true*) as a consequent.

Let $\Sigma = \{\alpha_1, ..., \alpha_n\}$ be a set of hypotheses. The formula α is defined to be a *deductive con quence* of Σ if either α is an *axiom* or a *hypothesis* or is *derived* from α_j, where $1 \le j \le n$, us modus ponen inference rule. It is represented as $\{\alpha_1, ..., \alpha_n\} \mid\!- \alpha$ or more formally as $\Sigma \mid\!- \alpha$. If is an empty set and α is deduced, then we can write $\mid\!- \alpha$. In this case, α is deduced from axio only and no hypotheses are used. In such situations, α is said to be a *theorem*. To illustrate t concepts clearly let us consider the following example:

Example 4.5 Establish that $A \to C$ is a deductive consequence of $\{A \to B, B \to C\}$, i.e., $\{A \to$ $B \to C\} \mid\!- (A \to C)$.

Solution We can prove the theorem as shown below in Table 4.9.

Table 4.9 Proof of the theorem $\{A \to B, B \to C\} \mid\!- (A \to C)$

Description	Formula	Comments
Theorem	$\{A \to B, B \to C\} \mid\!- (A \to C)$	*Prove*
Hypothesis 1	$A \to B$	1
Hypothesis 2	$B \to C$	2
Instance of Axiom 1	$(B \to C) \to [A \to (B \to C)]$	3
MP (2, 3)	$[A \to (B \to C)]$	4
Instance of Axiom 2	$[A \to (B \to C)] \to [(A \to B) \to (A \to C)]$	5
MP (4, 5)	$(A \to B) \to (A \to C)$	6
MP (1, 6)	$(A \to C)$	*Proved*

Hence, we can conclude that $A \to C$ is a deductive consequence of $\{A \to B, B \to C\}$.

Deduction Theorem Given that Σ is a set of hypotheses and α and β are well-formed formul If β is proved from $\{\Sigma \cup \alpha\}$, then according to the deduction theorem, $(\alpha \to \beta)$ is proved fron Alternatively, we can write $\{\Sigma \cup \alpha\} \mid\!- \beta$ implies $\Sigma \mid\!- (\alpha \to \beta)$.

Converse of Deduction Theorem The converse of the deduction theorem can be stated as: Given $\Sigma \vdash (\alpha \rightarrow \beta)$, then $\{\Sigma \cup \alpha\} \vdash \beta$ is proved.

Useful Tips

The following are some tips that will prove to be helpful in dealing with an axiomatic system:

- If α is given, then we can easily prove $\beta \rightarrow \alpha$ for any well-formed formulae α and β.
- If $\alpha \rightarrow \beta$ is to be proved, then include α in the set of hypotheses Σ and derive β from the set $\{\Sigma \cup \alpha\}$. Then, by using deduction theorem, we can conclude that $\alpha \rightarrow \beta$.

Example 4.6 Prove $\vdash \sim A \rightarrow (A \rightarrow B)$ by using deduction theorem.

Solution If we can prove $\{\sim A\} \vdash (A \rightarrow B)$ then using deduction theorem, we have proved $\vdash \sim A \rightarrow (A \rightarrow B)$. The proof is shown in Table 4.10.

Table 4.10 Proof of $\{\sim A\} \vdash (A \rightarrow B)$

Description	Formula	Comments
Theorem	$\{\sim A\} \vdash (A \rightarrow B)$	*Prove*
Hypothesis 1	$\sim A$	1
Instance of Axiom 1	$\sim A \rightarrow (\sim B \rightarrow \sim A)$	2
MP (1, 2)	$(\sim B \rightarrow \sim A)$	3
Instance of Axiom 3	$(\sim B \rightarrow \sim A) \rightarrow (A \rightarrow B)$	4
MP (3, 4)	$(A \rightarrow B)$	*Proved*

4.6 Semantic Tableau System in Propositional Logic

In both natural deduction and axiomatic systems, forward chaining approach is used for constructing proofs and derivations. In this approach, we start proofs or derivations from a given set of hypotheses or axioms. In axiomatic system, we often require a guess for the selection of appropriate axiom(s) in order to prove a theorem. Although the forward chaining approach is good for theoretical purposes, its implementation in derivations and proofs is difficult. Two other approaches may be used: *semantic tableau* and *resolution refutation* methods; in both cases, proofs follow backward chaining approach. In semantic tableau method, a set of rules are applied systematically on a formula or a set of formulae in order to establish consistency or inconsistency.

Semantic tableau is a binary tree which is constructed by using semantic tableau rules with a formula as a root. These rules and building proofs using this method are discussed in detail in the following subsection.

4.6.1 Semantic Tableau Rules

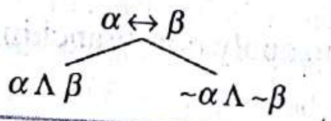
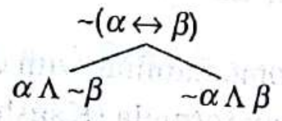
The semantic tableau rules are given in Table 4.11 where α and β are two formulae.

Table 4.11 Semantic Tableau Rules for α and β

Rule No.	Tableau tree	Explanation
Rule 1	*$\alpha \wedge \beta$ is true if both α and β are true* $$\begin{tikzcd} \alpha \wedge \beta \arrow[d, dash] \\ \alpha \arrow[d, dash] \\ \beta \end{tikzcd}$$	A tableau for a formula $(\alpha \wedge \beta)$ is constructed by adding both α and β to the same path (branch)
Rule 2	*$\sim(\alpha \wedge \beta)$ is true if either $\sim\alpha$ or $\sim\beta$ is true* $$\begin{tikzcd} & \sim(\alpha \wedge \beta) \arrow[dl, dash] \arrow[dr, dash] & \\ \sim\alpha & & \sim\beta \end{tikzcd}$$	A tableau for a formula $\sim(\alpha \wedge \beta)$ is constructed by adding two new paths: one containing $\sim\alpha$ and the other containing $\sim\beta$
Rule 3	*$\alpha \vee \beta$ is true if either α or β is true* $$\begin{tikzcd} & \alpha \vee \beta \arrow[dl, dash] \arrow[dr, dash] & \\ \alpha & & \beta \end{tikzcd}$$	A tableau for a formula $(\alpha \vee \beta)$ is constructed by adding two new paths: one containing α and the other containing β
Rule 4	*$\sim(\alpha \vee \beta)$ is true if both $\sim\alpha$ and $\sim\beta$ are true* $$\begin{tikzcd} \sim(\alpha \vee \beta) \arrow[d, dash] \\ \sim\alpha \arrow[d, dash] \\ \sim\beta \end{tikzcd}$$	A tableau for a formula $\sim(\alpha \vee \beta)$ is constructed by adding both $\sim\alpha$ and $\sim\beta$ to the same path
Rule 5	*$\sim(\sim\alpha)$ is true then α is true* $$\begin{tikzcd} \sim(\sim\alpha) \arrow[d, dash] \\ \alpha \end{tikzcd}$$	A tableau for $\sim(\sim\alpha)$ is constructed by adding α on the same path
Rule 6	*$\alpha \rightarrow \beta$ is true then $\sim\alpha \vee \beta$ is true* $$\begin{tikzcd} & \alpha \rightarrow \beta \arrow[dl, dash] \arrow[dr, dash] & \\ \sim\alpha & & \beta \end{tikzcd}$$	A tableau for a formula $\alpha \rightarrow \beta$ is constructed by adding two new paths: one containing $\sim\alpha$ and the other containing β
Rule 7	*$\sim(\alpha \rightarrow \beta)$ true then $\alpha \wedge \sim\beta$ is true* $$\begin{tikzcd} \sim(\alpha \rightarrow \beta) \arrow[d, dash] \\ \alpha \arrow[d, dash] \\ \sim\beta \end{tikzcd}$$	A tableau for a formula $\sim(\alpha \rightarrow \beta)$ is constructed by adding both α and $\sim\beta$ to the same path

(Co

Table 4.11 (Contd.)

Rule No.	Tableau tree	Explanation
Rule 8	$\alpha \leftrightarrow \beta$ *is true then* $(\alpha \wedge \beta) \vee (\sim \alpha \wedge \sim \beta)$ *is true*  $\alpha \leftrightarrow \beta$ $\alpha \wedge \beta$ $\sim\alpha \wedge \sim\beta$	A tableau for a formula $\alpha \leftrightarrow \beta$ is constructed by adding two new paths: one containing $\alpha \wedge \beta$ and other $\sim\alpha \wedge \sim\beta$ which are further expanded
Rule 9	$\sim(\alpha \leftrightarrow \beta)$ is *true* then $(\alpha \wedge \sim \beta) \vee (\sim \alpha \wedge \beta)$ *is true*  $\sim(\alpha \leftrightarrow \beta)$ $\alpha \wedge \sim\beta$ $\sim\alpha \wedge \beta$	A tableau for a formula $\sim(\alpha \leftrightarrow \beta)$ is constructed by adding two new paths: one containing $\alpha \wedge \sim\beta$ and the other $\sim\alpha \wedge \beta$ which are further expanded

Let us consider an example to illustrate the method of constructing semantic tableau for a formula. The convention used in this construction is self-explanatory. The first column consists of rule number applied on line number. The second column contains the derivation of semantic tableau and last column contains the line number.

Example 4.7 Construct a semantic tableau for a formula $(A \wedge \sim B) \wedge (\sim B \rightarrow C)$.

Solution The construction of the semantic tableau for the given formula $(A \wedge \sim B) \wedge (\sim B \rightarrow C)$ is shown in Table 4.12.

Table 4.12 Semantic Tableau for Example 4.7

Description	Formula	Line number
Tableau root	$(A \wedge \sim B) \wedge (\sim B \rightarrow C)$	1
Rule 1 (1)	$A \wedge \sim B$	2
	$\sim B \rightarrow C$	3
Rule 1 (2)	A	4
	$\sim B$	5
Rule 6 (3)	$\sim(\sim B)$ C	6
Rule 3 (6)	B √(open)	
	× (closed) $\{B, \sim B\}$	

Paths in a tableau tree extend from the root to the leaf nodes. There are two paths in the tree shown in Table 4.12 starting from the root to leaf nodes ending at B and C. It is observed that first path from root to B becomes closed because of the presence of complementary atoms B, ~B, while the other path remains open.

The thumb rule to construct a semantic tableau is to apply non-branching rules (such as rules 1 and 7) before branching rules.

4.6.2 Satisfiability and Unsatisfiability

Before we proceed any further, it is important to become familiar with certain terms that are u in the study of a tableau. For this, consider α to be any formula (Kaushik S 2002).

- A path is said to be *contradictory* or *closed* (finished) whenever complementary atoms appear on the same path of a semantic tableau. This denotes inconsistency.
- If all paths of a tableau for a given formula α are found to be closed, it is called a *contradictory tableau*. This indicates that there is no interpretation or model that satisfies α.
- A formula α is said to be *satisfiable* if a tableau with root α is not a contradictory tableau, that is, has at least one open path. We can obtain a model or an interpretation under which the formula α evaluated to be true by assigning T (true) to all atomic formulae appearing on the open path of semantic tableau of α.
- A formula α is said to be *unsatisfiable* if a tableau with root α is a contradictory tableau.
- If we obtain a contradictory tableau with root ~α, we say that the formula α is *tableau provable*. Alternatively, a formula α is said to be *tableau provable* (denoted by |- α) if a tableau with root ~ is a contradictory tableau.
- A set of formulae $S = \{\alpha_1, \alpha_2, ..., \alpha_n\}$ is said to *unsatisfiable* if a tableau with root $(\alpha_1 \wedge \alpha_2 \wedge ... \alpha_n)$ is a contradictory tableau.
- A set of formulae $S = \{\alpha_1, \alpha_2, ..., \alpha_n\}$ is said to be *satisfiable* if the formulae in a set are simultaneously true, that is, if a tableau for $\alpha_1 \wedge \alpha_2 \wedge ... \wedge \alpha_n$ has at least one open (or non-contradictory) path.
- Let S be a set of formulae. The formula α is said to be tableau provable from S (denoted by S |- α) if there is a contradictory tableau from S with ~α as a root.
- A formula α is said to be a *logical consequence* of a set S if and only if α is tableau provable from S.
- If α is tableau provable (|- α) then it is also valid (|= α) and vice versa.

Now we consider a few examples to illustrate the terminology discussed above.

Example 4.8 Show that a formula $\alpha : (A \wedge \sim B) \wedge (\sim B \rightarrow C)$ is satisfiable.

Solution The semantic tableau for $(A \wedge \sim B) \wedge (\sim B \rightarrow C)$ has been drawn in Table 4.12 and we observe that there are two paths in it of which one path is closed, while the other is open. This shows that the formula $(A \wedge \sim B) \wedge (\sim B \rightarrow C)$ is satisfiable. In order to find its model (that is, the interpretation under which the formula is true), we assign T (true) to all atomic formulae appearing on the open path. Therefore, $\{A = T; \sim B = T; C = T\}$ or alternatively $\{A = T; B = F; C = T\}$ is a model under which α is true. We can verify this using truth table approach also.

Example 4.9 Show that $\alpha : (A \wedge B) \wedge (B \rightarrow \sim A)$ is unsatisfiable using the tableau method.

Solution It can be proven that α: $(A \wedge B) \wedge (B \rightarrow \sim A)$ is unsatisfiable as shown in Table 4.13.

Table 4.13 Tableau Method for Example 4.9

Description	Formula	Line number
Tableau root	$(A \wedge B) \wedge (B \rightarrow \sim A)$	1
Rule 1 (1)	$A \wedge B$	2
	$B \rightarrow \sim A$	3
Rule 1 (2)	A	4
	B	5
Rule 6 (3)	$\sim B$ $\sim A$ $\times \{B, \sim B\}$ $\times \{A, \sim A\}$	

Example 4.10 Consider a set $S = \{\sim(A \vee B), (C \rightarrow B), (A \vee C)\}$ of formulae. Show that S is unsatisfiable.

Solution Consider the conjunction of formulae in the set as a root of semantic tableau. We see from Table 4.14 that such a tableau is contradictory; hence, S is unsatisfiable.

Table 4.14 Tableau Method For Example 4.10

Description	Formula	Line numb
Tableau root	$\sim(A \vee B) \wedge (C \rightarrow B) \wedge (A \vee C)$	1
Rule 1 (1)	$\sim(A \vee B)$	2
	$(C \rightarrow B)$	3
	$(A \vee C)$	4
Rule 4 (2)	$\sim A$	
	$\sim B$	
Rule 3 (4)	A C	
	$\times \{A, \sim A\}$	
Rule 6 (3)	$\sim C$ B	
	$\times \{C, \sim C\}$ $\times \{B, \sim B\}$	

Example 4.11 Show that a set $S = \{\sim(A \vee B), (B \rightarrow C), (A \vee C)\}$ is consistent.

Solution The set S can be shown to be consistent (Table 4.15) as follows:

Table 4.15 Tableau Method for Example 4.11

Description	Formula	Line numbe
Tableau root	$\sim(A \vee B) \wedge (B \rightarrow C) \wedge (A \vee C)$	1
Rule 1 (1)	$\sim(A \vee B)$	2
	$(B \rightarrow C)$	3
	$(A \vee C)$	4
Rule 4 (2)	$\sim A$	
	$\sim B$	
Rule 3 (4)	A C	
Rule 6 (3)	$\times \{A, \sim A\}$ $\sim B$ C	
	$\surd$ $\surd$	

Since the tableau of conjunction of formulae of S has open paths, S is satisfiable. Further, we can construct a model for S by assigning truth value T to each literal on open path, that is, $\{\sim A = T; \sim B = T; C = T\}$ or $\{A = F; B = F; C = T\}$.

Example 4.12 Show that B is a logical consequence of $S = \{A \rightarrow B, A\}$.

Solution Let us include $\sim B$ as a root with S in the tableau tree.

Table 4.16 Tableau Method for Example 4.12

Description	Formula	Line number
Tableau root	$\sim B$	1
Premise 1	$A \rightarrow B$	2
Premise 2	A	
Rule 6 (2)	$\sim A$ B	3
	$\times \{A, \sim A\}$ $\times \{B, \sim B\}$	

We see from Table 4.16 that B is tableau provable from S, that is, $\sim B$ as root gives contradictory tableau; thus B is a logical consequence of S.

Example 4.13 Show that $a : B \vee \sim(A \rightarrow B) \vee \sim A$ is valid.

Solution In order to show that α is valid, we have to show that α is tableau provable, that is, the tableau tree with $\sim\alpha$ is contradictory. Table 4.17 shows that α is a valid formula.

4.7 Resolution Refutation in Propositional Logic

Another simple method that can be used in propositional logic to prove a formula or derive a goal from a given set of clauses by contradiction is the *resolution refutation method*. The term *clause* is used to denote a special formula containing the boolean operators ~ and V. Any given formula can be easily converted into a set of clauses. The method to do this is explained later in this section. Resolution refutation is the most favoured method for developing computer-based systems that can be used to prove theorems automatically. It uses a single inference rule, which is known as *resolution based on modus ponen inference rule*. It is more efficient in comparison to NDS and Axiomatic system because in this case we do not need to guess which rule or axiom to apply in development of proofs. Here, the negation of the goal to be proved is added to the given set of clauses, and using the resolution principle, it is shown that there is a refutation in the new set. During resolution, we need to identify two clauses: one with a positive atom (P) and the o with a negative atom ($\sim P$) for the application of resolution rule. Before discussing resolutio clauses, let us desribe a method for converting a formula into a set of clauses.

Table 4.17 Tableau Method for Example 4.13

Description	Formula	Line number
Tableau root	$\sim(B \vee \sim(A \rightarrow B) \vee \sim A)$	1
Rule 4 (1)	$\sim B$	2
	$\sim[\sim(A \rightarrow B) \vee \sim A]$	3
Rule 4 (3)	$\sim[\sim(A \rightarrow B)]$	4
	$\sim(\sim A)$	5
Rule 5 (5)	A	6
Rule 5 (4)	$A \rightarrow B$	
Rule 6 (7)	$\sim A$ B	7
	$\times \{A, \sim A\}$ $\times \{B, \sim B\}$	

4.7.1 Conversion of a Formula into a Set of Clauses

In propositional logic, there are two normal forms, namely, *disjunctive normal form* (DNF) *conjunctive normal form* (CNF). A formula is said to be in its *normal form* if it is constru using only natural connectives $\{\sim, \wedge, \vee\}$. In DNF, the formula is represented as disjunctio conjunction, that is, in the form $(L_{11} \wedge \ldots \wedge L_{1m}) \vee \ldots \vee (L_{p1} \wedge \ldots \wedge L_{pk})$, whereas in CNF, represented as conjunction of disjunction, that is, in the form $(L_{11} \vee \ldots \vee L_{1m}) \wedge \ldots \wedge (L_{p1} \vee$ $L_{pk})$, where all L_{ij} are literals (positive or negative atoms). We can easily write the CNF form given formula as $(C_1 \wedge \ldots \wedge C_n)$, where each C_k $(1 \leq k \leq n)$ is a disjunction of literals and is ca a *clause*. Formally, a *clause* is defined as a formula of the form $(L_1 \vee \ldots \vee L_m)$. Therefore given formula is converted to its equivalent CNF as $(C_1 \wedge \ldots \wedge C_n)$, then the set of clauses is nothing but a set of each conjunct of CNF, that is, $\{C_1, \ldots, C_n\}$. For example, the set $\{A \vee B, \sim A \vee D, C \vee \sim B\}$ represents a set of clauses $A \vee B$, $\sim A \vee D$, and $C \vee \sim B$. The procedure by which such clauses for a given formula can be obtained is discussed in the following subsection.

4.7.2 Conversion of a Formula to its CNF

Any formula in propositional logic can be easily transformed into its equivalent CNF representation by using the equivalence laws described below. The following steps are taken to transform a formula to its equivalent CNF.

- Eliminate double negation signs by using

$$\sim(\sim A) \equiv A$$

- Use De Morgan's Laws to push ~ (negation) immediately before the atomic formula

$$\sim(A \wedge B) \equiv \sim A \vee \sim B$$
$$\sim(A \vee B) \equiv \sim A \wedge \sim B$$

- Use distributive law to get CNF

$$A \vee (B \wedge C) \equiv (A \vee B) \wedge (A \vee C)$$

- Eliminate $\rightarrow$ and $\leftrightarrow$ by using the following equivalence laws:

$$A \rightarrow B \equiv \sim A \vee B$$
$$A \leftrightarrow B \equiv (A \rightarrow B) \wedge (B \rightarrow A)$$

The method of conversion will become clearer with the help of the following examples:

Example 4.14 Convert the formula $(\sim A \rightarrow B) \wedge (C \wedge \sim A)$ into its equivalent CNF representation.

Solution The given formula $(\sim A \rightarrow B) \wedge (C \wedge \sim A)$ can be transformed into its CNF representation in the following manner:

$$(\sim A \rightarrow B) \wedge (C \wedge \sim A) \equiv (\sim(\sim A) \vee B) \wedge (C \wedge \sim A) \qquad \{\text{as } \sim A \rightarrow B \equiv (\sim A) \vee B\}$$
$$\equiv (A \vee B) \wedge (C \wedge \sim A) \qquad \{\text{as } \sim(\sim A) \equiv A\}$$
$$\equiv (A \vee B) \wedge C \wedge \sim A$$

The set of clauses in this case is written as $\{(A \vee B), C, \sim A\}$

4.7.3 Resolution of Clauses

Two clauses can be resolved by eliminating complementary pair of literals, if any, from bo new clause is constructed by disjunction of the remaining literals in both the clauses. Therefo two clauses C_1 and C_2 contain a complementary pair of literals $\{L, \sim L\}$, then these clauses ma resolved together by deleting L from C_1 and $\sim L$ from C_2 and constructing a new clause by disjunction of the remaining literals in C_1 and C_2. This new clause is called *resolvent* of C_1 and The clauses C_1 and C_2 are called *parent clauses* of the resolved clause. The resolution tree is inverted binary tree with the last node being a resolvent, which is generated as a part of resolution process. The process of resolution of clauses will become clearer with the help of following examples:

Example 4.15 Find resolvent of the clauses in the set $\{A \vee B, \sim A \vee D, C \vee \sim B\}$.

Solution The method of resolution is shown in Fig. 4.1.

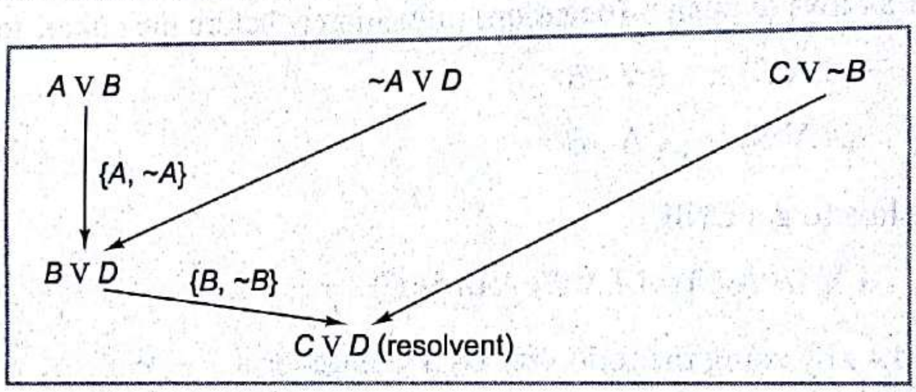


Figure 4.1 Resolution of clauses of Example 4.15

We can clearly see that $C \vee D$ is a resolvent of the set $\{A \vee B, \sim A \vee D, C \vee \sim B\}$.

Now that we are familiar with the resolution process, we can state a few results.

- If C is a resolvent of two clauses C_1 and C_2, then C is called a logical consequence of the set of t clauses $\{C_1, C_2\}$. This is known as *resolution principle*.
- If a contradiction (or an empty clause) is derived from a set S of clauses using resolution then S said to be *unsatisfiable*. Derivation of contradiction for a set S by resolution method is called *resolution refutation* of S.
- A clause C is said to be a *logical consequence* of S if C is derived from S.
- Alternatively, using the resolution refutation concept, a clause C is defined to be a *logical con quence* of S if and only if the set $S' = S \cup \{\sim C\}$ is unsatisfiable, that is, a contradiction (or an emp clause) is deduced from the set S', assuming that initially the set S is satisfiable.

Example 4.16 Using resolution refutation principle show that $C \vee D$ is a logical consequence of $S = \{A \vee B, \sim A \vee D, C \vee \sim B\}$.

Solution To prove the statement, first we will add negation of the logical consequence, that is, $\sim(C \vee D) \equiv \sim C \wedge \sim D$ to the set S to get $S' = \{A \vee B, \sim A \vee D, C \vee \sim B, \sim C, \sim D\}$. Now, we can show that S' is unsatisfiable by deriving contradiction using the resolution principle (Fig. 4.2).

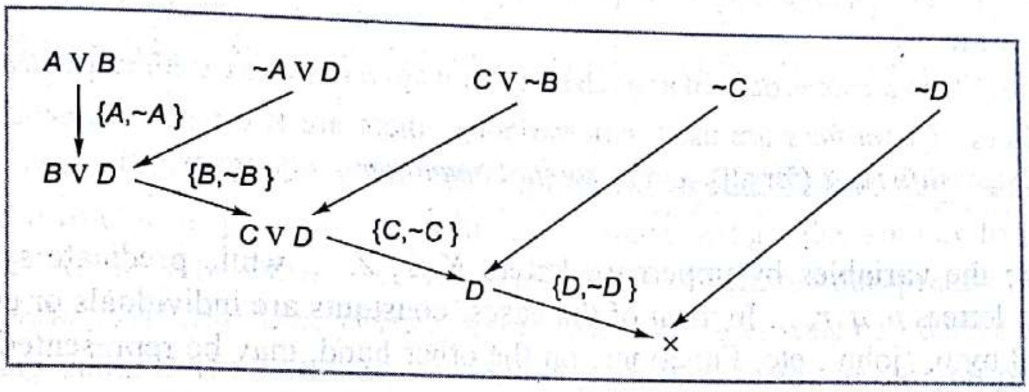


Figure 4.2 Resolution of Clauses for Example 4.16

Since we get contradiction from S', we can conclude that $(C \vee D)$ is a logical consequence of $S = \{A \vee B, \sim A \vee D, C \vee \sim B\}$.

4.8 Predicate Logic

In the preceding sections, we have discussed about propositional logic and various methods that can be used to show validity, unsatisfiability, etc., of a given proposition or a set of propositions. However, propositional logic has many limitations. For example, the facts *John is a boy*, *Paul is a boy*, and *Peter is a boy* can be symbolized by A, B, and C, respectively, in propositional logic but we can not draw any conclusions about the similarities between A, B, and C, that is, we cannot conclude that these symbols represent boys. Alternatively, if we represent these facts as boy(John), boy(Paul), and boy(Peter), then these statements give prima facie information that *John*, *Paul*, and *Peter* are all boys. These facts can be easily generated from a general statement boy(X), where the variable X is bound with *John*, *Paul*, or *Peter*. These facts are called instances of boy(X), while the statement boy(X) is called a *predicate statement or expression*. Here, *boy* is a predicate symbol and X is its argument. When a variable X gets bound to its actual value, then the predicate statement boy(X) becomes either *true* or *false*, for example, boy(Peter) = *true*, boy(Mary) = *false*, and so on.

Further, statements like *All birds fly* cannot be represented in propositional logic. Such limitations are removed in predicate logic. The predicate logic is a logical extension of propositional logic, which deals with the validity, satisfiability, and unsatisfiability (inconsistency) of a formula along with the inference rules for derivation of a new formula. Predicate calculus is the study of predicate systems; when inference rules are added to predicate calculus, it becomes predicate logic.

4.8.1 Predicate Calculus

Predicate calculus has three more logical notions in addition to propositional calculus. These described as follows:

- **Term** A *term* is defined as either a variable, or constant or n-place function. A *function* is defined as a mapping that maps n terms to a single term. An n-place function is written as $f(t_1, \ldots, t_n)$ where $t_1, \ldots, t_n$ are terms.
- **Predicate** A *predicate* is defined as a relation that maps n terms to a truth value {*true*, *false*}.
- **Quantifiers** *Quantifiers* are used with variables; there are two types of quantifiers, namely, *universal quantifiers*, $\forall$ (for all), and *existential quantifiers*, $\exists$ (there exists)

We can denote the variables by uppercase letters X, Y, Z, ... while predicate symbols can denoted by the letters p, q, r, ... In most of the cases, constants are individuals or concepts which can be denoted by 6, 'john', etc. Functions, on the other hand, may be represented by lowercase letters such as f, g, h, etc.

Well-formed formula In predicate calculus, well-formed formula (or simply formula) defined as follows:

- Atomic formula $p(t_1, \ldots, t_n)$ (also called an atom) is a well-formed formula, where p is a predicate symbol and $t_1, \ldots, t_n$ are the terms.
- If α and β are well-formed formulae, then $\sim(\alpha)$, $(\alpha \vee \beta)$, $(\alpha \wedge \beta)$, $(\alpha \rightarrow \beta)$, and $(\alpha \leftrightarrow \beta)$ are well-formed formulae.
- If α is a well-formed formula and X is a free variable in α, then $(\forall X)\alpha$ and $(\exists X)\alpha$ are both well-formed formulae. Here, α is in scope of quantifier $\forall$ or $\exists$. Scope of the variable X is defined as that part of an expression where all occurrences of X have the same value.
- Well-formed formulae may be generated by applying the rules described above a finite number of times.

The following are some examples of atomic formulae:

- A statement *X is brother of Y* can be easily represented in predicate calculus as brother(X, Y) which maps it to *true* or *false* when X and Y get instantiated to actual values. Here, brother is a predicate name.
- A statement *Peter loves his son* is represented as love("Peter", son("Peter")). Here, *son* is function that maps Peter to his son and *love* is a predicate name which takes two terms and maps them to *true* or *false* depending on the values of its terms.
- A statement *Every human is mortal* is translated into predicate calculus formula as follows: Let us represent the statement *X is a human* as human(X) and *X is mortal* as mortal(X). The corresponding formula can then be written as $(\forall X)$ (human(X) $\rightarrow$ mortal(X))

4.8.2 First-Order Predicate Calculus

If the quantification in predicate formula is only on simple variables and not on predicates or functions then it is called *first-order predicate calculus*. On the other hand, if the quantification is over first-order predicates and functions, then it becomes *second-order predicate calculus*. For example, $\forall p\ (p(X)) \leftrightarrow p(Y))$ is a second-order predicate statement, whereas $\forall X\ \forall Y\ (p(X) \leftrightarrow p(Y))$ is a first-order predicate statement. Similarly, higher order predicate calculus allows quantification over higher types. Since we will not be dealing with higher order predicates presently, we need not go into the details.

The first-order predicate calculus is a formal language in which a wide variety of statements are expressed. The formulae in predicate calculus are formed using rules similar to those used in propositional calculus. When inference rules are added to first-order predicate calculus, it becomes *first-order predicate logic* (FOL). Using inference rules, one can derive new formulae from the existing ones. In the following subsection we will describe interpretation of predicate formula which is quite different from interpretation of propositional formula.

4.8.3 Interpretations of Formulae in FOL

In propositional logic, an *interpretation* simply refers to an assignment of truth values to atoms. Since variables are involved in FOL, we need to do more than a simple assignment of values. An interpretation of a formula α in FOL is not just restricted to assigning truth values; it consists of a non-empty domain D and involves assignment of values to each constant, function symbol and also an assignment of truth values to each predicate atom. Each formula α is evaluated to be *true* or *false* under a given interpretation I over a given domain D.

The following results hold true for any interpretation I over a domain D:

- $(\forall X)\ p(X) = $ *true* if and only if $p(X) = $ *true*, $\forall X \in D$ otherwise it is *false*.
- $(\exists X)\ p(X)] = $ *true* if and only if $\exists\ c \in D$ such that $p(c) = $ *true*, otherwise it is *false*.

Example 4.17

(i) Evaluate the truth value of an FOL formula α: $(\forall X)\ (\exists Y)\ p(X, Y)$ under the following interpretation I:

- $D = \{1, 2\}$
- $p(1, 1) = F, p(1, 2) = T, p(2, 1) = T,\ p(2, 2) = F$

Solution Let us denote *true* by T and *false* by F. For $X = 1$, then $\exists\ 2 \in D$ such that $p(1, 2) = T$ and for $X = 2$, then $\exists\ 1 \in D$ such that $p(2, 1) = T$. Hence, α is *true* under interpretation I.

(ii) Evaluate $\alpha: (\forall X)\,[p(X) \rightarrow q(f(X), c)]$ under the following interpretation:

- $D = \{1, 2\}$
- $c = 1$ (c is a constant from the domain D)
- $f(1) = 2, f(2) = 1$
- $p(1) = F, p(2) = T$
- $q(1, 1) = T, q(1, 2) = T, q(2, 1) = F, q(2, 2) = T$

Solution For $X = 1$

$$p(1) \rightarrow q(f(1), 1) \equiv p(1) \rightarrow q(2, 1) \equiv F \rightarrow q(2, 1) \equiv T$$

For $X = 2$

$$p(2) \rightarrow q(f(2), 1) \equiv p(2) \rightarrow q(1, 1) \equiv T \rightarrow q(1, 1) \equiv T \rightarrow T = T$$

We can easily say that α is true for all values of $X \in D$ under the interpretation I.

4.8.4 Satisfiability and Unsatisfiability in FOL

To study FOL in detail we need to be familiar with the following definitions for a given formula

- A formula α is said to be *satisfiable* if and only if there exists an interpretation I such that α evaluated to be *true* under I. Alternatively, we may say that I is a *model* of α or I satisfies α.
- A formula α is said to be *unsatisfiable* if and only if $\exists$ no interpretation that satisfies α or $\exists$ no model for α.
- A formula α is said to be *valid* if and only if for every interpretation I, α is *true*.
- A formula α is called a *logical consequence* of a set of formulae $\{\alpha_1, \alpha_2, ..., \alpha_n\}$ if and only if for every interpretation I, if $\alpha_1 \wedge ... \wedge \alpha_n$ is evaluated to be *true*, then α is also evaluated to be *true*.

Since there are finite interpretations in propositional logic, it is possible to verify validity, satisfiability, and unsatisfiability of a formula; however, in FOL, there are infinite number domains and consequently infinite number of interpretations of a formula. Therefore, it is possible to verify validity and unsatisfiability of a formula by evaluating it under infinite interpretations. We can easily solve this problem in predicate logic by using resolution refutation meth which is similar to propositional logic. In this method, we work with clauses similar propositional logic; however, obtaining clauses in FOL is not as straightforward as it is in the c of propositional logic, where we convert a given formula to its equivalent CNF notation, conjunct of which is a clause. In addition to DNF and CNF notations, there is another nota called the *prenex normal form* (PNF), which is used for obtaining clauses from an FOL form

4.8.5 Transformation of a Formula into Prenex Normal Form

A formula is said to be in *closed form* if all the variables appearing in it are quantified and there are no free variables.

Prenex Normal Form A closed formula α in FOL is said to be in PNF if and only if α is represented as $(Q_1 X_1)(Q_2 X_2) \ldots (Q_n X_n) M$, where Q_k are quantifiers ($\forall$ or $\exists$), X_k are variables, for $1 \leq k \leq n$, while M is a formula free from quantifiers. The list of quantifiers $[(Q_1 X_1) \ldots (Q_n X_n)]$ is called *prefix* and M is called the *matrix* of a formula α. Here, M is assumed to be represented in CNF notation. For example, $(\exists X)(\forall Y)[p(X) \vee q(X, Y)]$ is in PNF notation, whereas $(\forall X)[p(X) \rightarrow (\exists Y)\, q(X, Y)]$ is not in PNF notation.

Conversion of Formula into PNF Notation

A formula can be easily transformed or converted into PNF using various equivalence laws. The following are some of the conventions that will be used to understand the concept clearly:

- α — FOL formula α without a variable X
- $\alpha[X]$ — FOL formula α which contains a variable X
- Q — Quantifier ($\forall$ or $\exists$).

Equivalence Laws

Although a number of equivalence laws of propositional logic have been studied earlier, the following pairs of logically equivalent formulae need to be discussed in addition. Here, the symbol * represents Λ or V.

Law 1 $(QX)\, \alpha[X] * \beta \equiv (QX)(\alpha[X] * \beta)$

Law 2 $\alpha * (QX)\, \beta[X] \equiv (QX)(\alpha * \beta[X])$

Law 3 $\sim(\forall X)\, \alpha[X] \equiv (\exists X)(\sim\alpha[X])$

Law 4 $\sim(\exists X)\, \alpha[X] \equiv (\forall X)(\sim\alpha[X])$

Any formula that does not contain X can be brought into the scope of the quantifier Q on X. Therefore, the first two equivalences hold true; these equivalences can be easily proved. Let us prove the last equivalence for the sake of clarity.

Result $\sim(\exists X)\, \alpha[X] \equiv (\forall X)(\sim\alpha[X])$

Proof Let I be any interpretation over a domain D. To prove the equivalence law $\sim(\exists X)\alpha[X] \equiv (\forall X)(\sim\alpha[X])$, we have to prove that $\sim(\exists X)\, \alpha[X]$ is *true* if and only if $(\forall X)(\sim\alpha[X])$ is *true* under any interpretation I over any domain D.

Let us assume that $\sim(\exists X)\, \alpha[X]$ is *true* under I over D and prove that $(\forall X)(\sim\alpha[X])$ is also *true* under I over D. Since $\sim(\exists X)\, \alpha[X]$ is *true* (assumption), it implies that there does not exist any X for which $\alpha[X]$ is *true*, that is, for all X, $\alpha[X]$ is *false*.

- A formula α is said to be *unsatisfiable* if and only if its corresponding set S is unsatisfiable.
- S is said to be *unsatisfiable* if and only if there $\exists$ no interpretation that satisfies all the clauses o simultaneously.
- S is said to be *satisfiable* if and only if each clause is satisfiable, i.e., $\exists$ an interpretation that satis all the clauses of S simultaneously.
- Alternatively, an interpretation I is said to model S if and only if I models each clause of S.

4.8.8 Resolution Refutation Method in FOL

Resolution refutation method in FOL is used to test unsatisfiability of a set (S) of cla corresponding to the predicate formula. It is an extension of the resolution refutation me described earlier for propositional logic. A deduction of a contradiction from a set S of clau called a *resolution refutation* of S. The resolution principle basically checks whether a contra tion is contained in or derived from S.

Resolution for the clauses containing no variables is simple and is similar to that use propositional logic, but it becomes complicated when clauses contain variables. In such ca before resolution, two complementary literals are resolved after proper substitutions so that the literals have same arguments. Let us consider the following example:

Example 4.19 Find the resolvent of two clauses CL_1 and CL_2, where p, q, and r are predi symbols, X is a variable and f is a unary function.

$$CL_1 = p(X) \vee q(X)$$
$$CL_2 = \sim p(f(X)) \vee r(X)$$

Solution If we substitute $f(a)$ for X in CL_1 and a for X in CL_2, where a is a new constant from the do then we obtain

$$CL_3 = p(f(a)) \vee q(f(a))$$
$$CL_4 = \sim p(f(a)) \vee r(a)$$

We observe that clauses CL_3 and CL_4 have complementary literals as $p(f(a))$ and $\sim p(f(a))$. The resolv generated by taking the disjunction of literals from parent clauses after eliminating pair of compleme literals. Therefore, we get resolvent of CL_3, and CL_4 as $CL = q(f(a)) \vee r(a)$.

Here we notice that CL_3 and CL_4 do not have variables. These are called ground instances of CL_1 and Cl general, if we substitute $f(X)$ for X in CL_1, then we get another clause as follows:

$$CL_1' = p(f(X)) \vee q(f(X))$$

Then, $CL' = \text{Resolvent}(CL_1', CL_2) = [q(f(X)) \vee r(X)]$

- As the first step, choose a clause from the negated goal clauses as one of the parents to be resolv this is done because the contradiction, if it exists, might be occurring due to the goal we are tryin prove.
- Use the resolvent for further resolution with an existing clause. Both the clauses sho contain one pair of complementary literals, if possible. If parent clauses contain more than one p of complementary literals, then resolvent is always true.
- If such clauses do not exist, then resolve any pair of clauses that may contain complement literals.
- Resolve using clauses with a single literal whenever possible. Such resolutions generate new clau with fewer literals than the larger of their parent clauses; thus, the algorithm may most prob terminate faster.
- Eliminate tautologies as soon as they are generated.

Example 4.20 Show that the formula $\alpha : (\forall X)\ (p(X) \wedge \sim[q(X) \rightarrow p(X)])$ is unsatisfiable.

Solution To prove the above statement, we need to convert α into a set of clauses with the hel equivalence laws.

$$\begin{aligned} p(X) \wedge \sim[q(X) \rightarrow p(X)] &\cong p(X) \wedge \sim[\sim q(X) \vee p(X)] \\ &\cong p(X) \wedge \sim\sim q(X) \wedge \sim p(X) \\ &\cong p(X) \wedge q(X) \wedge \sim p(X) \end{aligned}$$

The set of clauses is written as $S = \{p(X), q(X), \sim p(X)\}$. Since there is a contradiction in S itself [becaus $p(X)$ and $\sim p(X)$], S is unsatisfiable and consequently α is unsatisfiable.