University of El Oued

Computer Science Department

Level: License 1

Subject: Mathematical Logic

Learning Objectives: Formalization of human reasoning

Prior Knowledge: Basic knowledge of mathematics and Boolean algebra

Mathematical Logic: Foundations of Propositional and Predicate Reasoning

course materials & exercises

Version 1
Year: 2025/2026

Dr. Mostefa Kara
karamostefa@univ-eloued.dz

General Introduction

Logic is the calculus of computer science. Just as physics is built upon classical mechanics, the digital world, from the simplest circuit to the most complex artificial intelligence, is built upon the formal rules of logic. This course provides a rigorous journey through the two primary pillars of logic: propositional logic and predicate logic.

This course provides a structured and progressive introduction to mathematical logic, with a particular focus on propositional logic and predicate logic. It is designed to equip students with the theoretical foundations and formal reasoning skills necessary for rigorous analysis in computer science (CS) and related disciplines. The course begins with fundamental logical concepts and gradually advances toward more expressive logical systems, ensuring a smooth transition from basic notions to formal methods of reasoning.

The content is organized into twelve sessions (1h30 each), every session corresponding to a specific set of learning objectives and practical exercises (TDs 1-12). Topics are presented in a logical sequence. In the first half of our study, we focus on syntax and semantics, learning how to construct well-formed formulas and evaluate their truth through interpretations and truth tables. We then transition into automated reasoning, exploring how computers use the resolution method to derive conclusions through mechanical refutation.

Finally, we expand our language into First-Order Logic (Predicate Logic). Here, we move beyond simple statements to describe a universe of objects, relations, and quantities. By the end of these twelve sessions, you will have developed the analytical toolkit necessary to verify software correctness, design database queries, and understand the fundamental limits of what can be computed. This organization ensures conceptual continuity and reinforces understanding through progressive abstraction.

Each session is complemented by a corresponding tutorial (TD) that allows students to apply theoretical concepts through guided exercises. This integrated structure promotes active learning, strengthens problem-solving skills, and supports the development of formal reasoning competencies essential for advanced studies and professional practice.

Course Content

Content by Sessions

Following

1 Session I: Foundations and Objects of Mathematical Logic

1.1 Introduction

Logic is a fundamental pillar of Computer Science, serving as the calculus of computer science. It provides the formal tools needed to express ideas precisely, reason correctly, and verify whether conclusions follow from given information.

In everyday life, humans reason informally using natural language. However, natural language is often ambiguous, imprecise, and context-dependent. Logic replaces this informal reasoning with Formal Systems governed by strict rules. The objective is to transform intuitive arguments into precise mathematical objects that can be analyzed rigorously.

What is Logic? Logic is the branch of mathematics and philosophy that studies the principles of valid reasoning. It is defined as the study of the methods used to distinguish correct from incorrect reasoning, and it is prescriptive rather than descriptive: logic does not investigate how people actually think, as psychology does, but instead focuses on how one ought to reason in order to reach the truth. Moreover, logic is concerned with the form of reasoning rather than its specific subject matter or content, allowing the same logical principles to be applied across different domains.

Intuitive Reasoning: Fast, based on experience, feelings, or common sense. However, it is often flawed. Example: “All expensive cars are fast. This car is fast, so it must be expensive.” (Intuitively tempting but logically invalid).

Logical Reasoning: Slow and systematic. It is based on strict rules of inference and is guaranteed to be correct if those rules are followed. Example: “If it is a square, it has four sides. This is a square. Therefore, it has four sides.” (Logically valid).

1.2 The Objects of Logic

To build a system of logic, we must define the main objects of study: Propositions, Truth Values, Arguments, Inference, and Validity.

The Proposition (The Atom): A proposition, also called an atom, is a declarative statement that is either True or False, but not both. This definition relies on the Principle of Bivalence, also known as the Law of Excluded Middle, which states that every proposition must have exactly one truth value, either 1 or 0, with no intermediate possibility. In addition, the Law of Non-Contradiction ensures that a proposition cannot be simultaneously True and False at the same time.

Examples of Propositions:

- ✓ “$2 + 2 = 4$” (True)
- ✓ “5 is an even number” (False)
- ✓ “Paris is the capital of France” (True)
- ✓ “7 is prime” (True)

Non-Examples (Not Propositions):

- Commands: “Close the door.” (No truth value).
- Questions: “Is it raining?” (Cannot be true or false).
- Open Sentences: “$x + 1 > 3$” (Value depends on x; becomes a proposition only when x is defined).

Truth Values: The truth value of a proposition is its status as True (T/1) or False (F/0).

The Argument (The Molecule): An argument is a sequence or collection of propositions consisting of:

Premises: Propositions offered as evidence, assumptions, or reasons.

Conclusion: The proposition claimed to follow logically from the premises.

Structure of an Argument:

$$\begin{array}{l} \text{Premise 1} \\ \text{Premise 2} \\ \hline \text{Conclusion} \end{array}$$

Standard Example:

$$\begin{array}{l}\text{Premise 1: All humans are mortal.}\\ \text{Premise 2: Socrates is a human.}\\ \hline \text{Conclusion: Socrates is mortal.}\end{array}$$

Inference (The Process): Inference is the linking step, the movement of the mind from premises to the conclusion using rules of reasoning.

- Informal Inference: Based on intuition, language, and common sense. Example: "It is raining, so the ground is wet."
- Formal Inference: Based on symbols and strict mathematical rules. CS relies exclusively on formal inference so that reasoning can be checked, automated, and implemented.
  Example:
  $$P \to Q, \quad P \ \vdash \ Q$$
  (Modus Ponens)

1.3 Validity and Truth

This distinction is crucial for rigorous logic.
Truth: A property of individual propositions (factual correctness).
Validity: A property of arguments. An argument is valid if: *whenever all premises are true, the conclusion must also be true.*

Important Note: An argument can be valid with false premises, or invalid with a true conclusion. Logic studies the validity of the structure, not the factual truth.
Example of Valid but False Content:

Premise: All birds can fly. (False)
Premise: Penguins are birds. (True)
Conclusion: Penguins can fly. (False)

The reasoning is perfect (Valid), even though the content is factually incorrect.

1.4 Syntax and Semantics

Syntax (The Grammar): The rules for constructing formulas and strings of symbols.
Analogy: In Python, `if x > 5:` is syntactically correct. `if > 5 x` is a syntax error.
Semantics (The Truth): The meaning or truth value of those formulas.
Analogy: `if 10 > 5:` is semantically True. `if 2 > 5:` is semantically False.

1.5 The Role of Logic in Computer Science

Formal logic allows us to eliminate ambiguity and to verify correctness in a mechanical and rigorous way, making it foundational to Computer Science. It underpins digital circuit design, where hardware transistors operate according to Boolean logic with values 0 and 1, and it is essential for program specification and verification, enabling critical software systems such as avionics and medical devices to be proven correct rather than merely tested. Logic also lies at the heart of algorithm design, since every if-then-else construct is a logical conditional, and it forms the basis of database theory, where SQL queries rely on predicate logic. Furthermore, formal logic is central to artificial intelligence through knowledge representation and automated reasoning, and to cybersecurity, where security protocols and cryptographic schemes are validated using formal logical proofs.

1.6 Summary

In this session, we transformed human reasoning from informal "gut feelings" into formal mathematical objects. By understanding Propositions, Arguments, and the difference between Truth and Validity, we establish the foundation for the entire study of logic in Computer Science.

Go to Exercises

2 Session II: Syntax, Semantics, and Formalization

2.1 Motivation and Introduction

In the previous session, we introduced logic as a formal study of reasoning. However, to reason formally, we must transition from the “what” of logic to the “how.” This requires answering a fundamental question: How do we precisely describe expressions and their meaning?

This leads to a crucial distinction in logic and Computer Science:

Syntax $\rightarrow$ The form and structure of expressions.
Semantics $\rightarrow$ The meaning and truth value of expressions.

This distinction is central not only in logic but also in programming languages, compilers, formal verification, and database systems.

2.2 Informal and Formal Languages

Natural Language
Natural languages such as English, French, or Arabic are rich, flexible, and highly expressive, but they are poorly suited for formal reasoning. They are inherently ambiguous, since a single sentence can admit multiple interpretations; for example, the statement “I saw the man with the telescope” does not clearly specify whether the man or the speaker has the telescope. Their meaning is also strongly context-dependent, varying with the speaker, location, or time, as in expressions like “this” or “here”. Moreover, natural languages are often imprecise, relying on vague definitions or unstated assumptions, which makes them unsuitable for rigorous logical analysis.

The Need for Formal Languages: To reason rigorously and to represent reasoning without the risk of interpretation errors, we rely on formal languages. A formal language is based on a fixed alphabet, meaning it consists of a finite and well-defined set of symbols. These symbols are combined according to strict formation rules that are mathematically precise, ensuring that only well-formed expressions are constructed. Consequently, every expression in a formal language has an unambiguous meaning, with exactly one valid interpretation. In addition, formal languages are computable, which means that their expressions can be processed by machines mechanically, without any need for cultural understanding, intuition, or contextual interpretation.

2.3 Syntax: The Anatomy of a Formula

Syntax is the part of logic that studies the form and structure of expressions, independently of their meaning. It answers questions such as: Is this expression well-formed? or Is it written correctly according to the rules?

The Alphabet of Propositional Logic: Every formal language begins with an alphabet (a finite set of symbols). In logic, this includes:

- Propositional Variables: Symbols like p, q, r, s or P, Q, R, S (Representing atomic facts).
- Logical Connectives/Symbols:
  $\neg$ (Unary): Negation.
  $\wedge, \vee, \Rightarrow, \Leftrightarrow$ (Binary): Connect two formulas ($\rightarrow, \leftrightarrow$ may also be used).
- Auxiliary Symbols: Parentheses “(” and “)”.

The logical connectives are read as follows:
$\neg P$: read as “not P”. It denotes the negation of P.
$P \wedge Q$: read as “P and Q”. It denotes the conjunction of P and Q.
$P \vee Q$: read as “P or Q”. It denotes the disjunction of P and Q.
$P \rightarrow Q$: can be read as “P implies Q”, “if P then Q”, “Q if P”, or “P only if Q”. It denotes the implication from P to Q.
$P \leftrightarrow Q$: read as “P if and only if Q”. It denotes the equivalence between P and Q.

Formation Rules (The Grammar): Formation rules specify how symbols can be combined to form valid expressions. A string of symbols that follows these rules is called a Well-Formed Formula (WFF):

Rule 1: Any propositional variable is a WFF.
Rule 2: If A is a WFF, then $(\neg A)$ is a WFF.
Rule 3: If A and B are WFFs, then $(A \wedge B), (A \vee B), (A \Rightarrow B)$, and $(A \Leftrightarrow B)$ are WFFs.
Rule 4: Nothing else is a WFF.
Examples (Syntax Only): Assume p and q are propositional variables.
✓ $(p \wedge q) \Rightarrow r \rightarrow$ syntactically correct.
✓ $\neg p \rightarrow$ syntactically correct.
X $\wedge pq \rightarrow$ syntactically incorrect.
X $(p \neg q) \rightarrow$ syntactically incorrect.
X $p \wedge \vee q \rightarrow$ syntactically incorrect.

Important Note: At this stage, meaning does not matter. Syntax only defines what can be written.

2.4 Semantics: Bringing Formulas to Life

Semantics is the part of logic that studies the meaning of syntactically correct expressions. It answers: Is this formula true or false? And: Under which conditions is it true?

Interpretation (Valuation): To give meaning to a formula, we need an interpretation. In propositional logic, an Interpretation assigns a truth value (T or F) to each propositional variable.
Example: If interpretation $p = T$ and $q = F$, the formula $(p \wedge q)$ results in $(T \wedge F) = F$.

Semantic Rules for Connectives (Semantic Evaluation): Once an interpretation is fixed, the truth value of any formula is determined by its parts:

$v(\neg A) = 1$ if $v(A) = 0$, and vice versa.
$v(A \wedge B) = 1$ only if both are 1.
$v(A \Rightarrow B) = 0$ only if $v(A) = 1$ and $v(B) = 0$.

Crucial: An implication is only false when a true premise leads to a false conclusion.

Syntax and Semantics: Fundamental Distinction: A formula can be syntactically correct but semantically false (e.g., $p \wedge \neg p$), or semantically true under one interpretation and false under another.

Aspect	Syntax	Semantics
Focus	Form / Structure	Meaning / Truth
Core Question	Is it well-written?	Is it true?
Depends on Interpretation?	No	Yes
Programming Analogy	Syntax error: Missing semicolon; Wrong keyword	Semantic error: Division by zero; Incorrect logic (Infinite loop)

The Importance of Formalization: Formalization is the process of translating a problem from natural language into a formal language. This step is crucial because it introduces precision by eliminating noise and ambiguity, leaving only the underlying logical structure of the problem. Once a statement is formalized, it becomes possible to verify its correctness using mechanical algorithms such as truth tables or resolution, enabling the construction of formal proofs. Furthermore, formalization makes automation possible: while computers cannot reason directly with natural language, they can operate on bits, symbols, and logic gates. This capability forms the foundation of parsers, compilers, and automated reasoning systems in artificial intelligence.

2.5 Summary

Understanding syntax and semantics allows us to design formal languages, build compilers, and verify complex systems. This distinction remains central throughout the course: syntax defines what can be written, while semantics defines what it means.

Go to Exercises

3 Session III: Propositional Logic - Syntax (Part 1)

3.1 Introduction to Propositional Logic

Propositional Logic (PL) is the simplest and most fundamental logical system. It moves us from the general philosophy of reasoning into a specific, symbolic mechanics. PL focuses on how propositions, the basic building blocks of thought, can be combined using logical operators to form more complex statements. In this session, we transition from informal human reasoning to a rigorous symbolic language.

3.2 Propositional Building Blocks

In PL, we treat propositions as the smallest units of information. We categorize them based on their complexity.

a. Atomic Propositions (Atoms): An atomic proposition is a simple declarative statement that contains no logical connectives. It cannot be broken down further without losing its meaning. In formal logic, these are represented by propositional variables (typically uppercase or lowercase letters).

Example 1 (p): "The server is active."
Example 2 (q): "The database is offline."
Example 3 (r): "The input is valid."

b. Compound Propositions: A compound proposition is formed by combining one or more atomic propositions using logical connectives. Its truth value depends entirely on the truth values of its component atoms.
Example: "The server is active, and the database is offline" ($p \wedge q$).

3.3 Logical Connectives

Connectives are the operators of logic. In computer science, they function similarly to arithmetic operators ($+, -, \times$), but they operate on truth values (Boolean values).

Negation
Symbol:$\neg$
Role: A unary operator (affects only one formula). It reverses the truth value.
Natural Language: "not", "it is false that", "it is not the case that".
Truth Table:

p	$\neg p$
T	F
F	T

Conjunction
Symbol: $\wedge$
Role: A binary operator. It is true only if both components are true.
Natural Language: "and", "but", "moreover", "while", "although".
Note: In logic, "but" is treated the same as "and" because both imply that both facts are simultaneously true.

Disjunction
Symbol: $\vee$
Role: A binary operator. It is true if at least one component is true.
Natural Language: "or", "unless".
Critical Distinction: This is the Inclusive OR (A or B or both). In computer science, this differs from the Exclusive OR (XOR), which is false if both are true.

Material Implication
Symbol: $\Rightarrow$ or $\rightarrow$
Role: Represents a conditional relationship.
Structure: In $p \Rightarrow q$, p is the Antecedent (Hypothesis) and q is the Consequent (Conclusion).
Semantic Rule: Crucial: The implication is false only when the antecedent (p) is true, and the consequent (q) is false.
CS Context: This is the logic behind `if (condition) { action }`.

Biconditional / Equivalence
Symbol: $\Leftrightarrow$ or $\leftrightarrow$
Role: True only if both components have the same truth value.
Natural Language: "if and only if" (iff), is equivalent to "just in case".

3.4 The Translation Map (Natural Language to Logic)

One of the most challenging tasks is mapping human speech to logical symbols.

Connective	Natural Language Cues	Logical Form
Negation	"It is not true that...", "None", "Never"	$\neg p$
Conjunction	"p although q", "p but q", "p while q"	$p \wedge q$
Disjunction	"p or q (or both)", "p unless q"	$p \vee q$
Implication	"q if p", "p is sufficient for q", "p only if q"	$p \Rightarrow q$
Equivalence	"p is necessary and sufficient for q"	$p \Leftrightarrow q$

Enriched Explanation: The "If" and "Only If" Trap: Students often struggle with the direction of the arrow.
"A if B": This means B is the condition for A. Translation: $B \Rightarrow A$.
"A only if B": This means A cannot be true without B being true. Translation: $A \Rightarrow B$.
Sufficient vs. Necessary: If p is sufficient for q, it means $p \Rightarrow q$. If p is necessary for q, it means $q \Rightarrow p$.

Warning: Reversing the direction of implication is the most common error in logic exams!

Importance in Computer Science: Logical connectives are not merely theoretical constructs; they play a central role in practical CS applications. In programming, control flow structures such as `if`, `while`, and `switch` statements depend on evaluated Boolean expressions to determine the execution path of a program. At the hardware level, digital circuits are built from logical gates like AND, OR, NOT, and NAND, which form the physical foundation of CPU operations. Logical connectives are also fundamental to Boolean algebra, which is used in code optimization to simplify complex conditional expressions and reduce the number of checks a processor must perform. Additionally, logic is essential in program verification, where formal methods are applied to prove that a program correctly satisfies its specified requirements.

3.5 Summary

In Session III, we defined the core components of propositional logic. We introduced atomic and compound propositions, where atomic propositions serve as the basic units of meaning and compound propositions are formed by combining them. We also covered the five main logical connectives: negation, conjunction, disjunction, implication, and biconditional. Finally, we developed translation skills for converting complex natural language statements, including "if/then" and "only if" constructions, into precise symbolic formulas.

Go to Exercises

4 Session IV: Variables, Well-Formed Formulas, and Substitution

4.1 Introduction and Motivation

In previous sessions, we explored the vocabulary of logic. Session IV transitions us into the grammar and algebra of logic. To reason formally and systematically, we cannot rely on long natural-language sentences. Instead, we introduce propositional variables to represent atomic facts, formal syntactic rules to construct well-formed logical expressions, and substitution as a mechanism for manipulating and transforming formulas. Together, these concepts form the bedrock of programming language theory, symbolic computation, and automated theorem proving.

4.2 Propositional Building Blocks: Variables and Constants

In algebra, we use x and y to represent unknown numbers. In logic, we use propositional variables to represent unknown truth values.

Propositional Variables
Definition: A symbol (usually p, q, r, s or $p_1, p_2, \dots$) that stands for an arbitrary atomic proposition.
Domain: The domain of these variables is strictly the set $\{True, False\}$ (or $\{1, 0\}$).
Purpose: Variables allow us to manipulate propositions symbolically and discuss the form of an argument regardless of its specific content.
Example: Let p: "The user is authenticated," q: "Access is granted," and r: "The system logs the event." We can now discuss the relationship between $p, q,$ and r mathematically.

Logical Constants (Verum and Falsum): Formal logic often includes two special constant symbols that represent fixed values:
$\top$ (Top / Verum): A constant that isalways True.
$\perp$ (Bottom / Falsum): A constant that is always False.

4.3 Well-Formed Formulas (WFF)

A Well-Formed Formula is a string of symbols that is grammatically correct according to the syntax of Propositional Logic. This is analogous to a Syntax Check performed by a compiler; if an expression is not a WFF, it has no logical meaning.

Recursive Definition of a WFF: To determine if a formula is Well-Formed, we apply these four strict rules:

1. Base Case (Atomic Formulas): Every propositional variable or constant is a WFF.
2. Negation Rule: If ϕ is a WFF, then $(\neg\phi)$ is a WFF.
3. Binary Rule: If ϕ and ψ are WFFs, then $(\phi \wedge \psi), (\phi \vee \psi), (\phi \Rightarrow \psi)$, and $(\phi \Leftrightarrow \psi)$ are WFFs.
4. Closure Rule: Nothing else is a WFF.

The Importance of the Recursive Structure: This recursive definition ensures that every formula has a finite construction and a unique, clear structure. It eliminates ambiguity, allowing a computer to parse the formula as a syntax tree.

Syntactic Examples

✓ p
✓ $\neg p$
✓ $(p \wedge (q \Rightarrow r))$ (Variables: p, q, r; Well-formed: Yes)
X $p\wedge$ (Missing second operand)
X $(\Rightarrow pq)$ (Incorrect operator placement)
X $(pq \wedge r)$ (Syntax error: variables must be separated by connectives)

4.4 Parentheses and Precedence

In logic, parentheses define the scope of a connective. However, to make formulas more readable, we use a Priority Hierarchy (similar to PEMDAS in mathematics).

Precedence Rules (Highest to Lowest):

1. $\neg$ (Negation)
2. $\wedge$ (Conjunction)
3. $\vee$ (Disjunction)
4. $\Rightarrow$ (Implication)
5. $\Leftrightarrow$ (Biconditional)

Example: $\neg p \wedge q$ is interpreted as $(\neg p) \wedge q$, whereas $\neg(p \wedge q)$ negates the entire conjunction.

4.5 Substitution in a Formula

Substitution is the process of replacing a propositional variable in a formula with another formula.

The Substitution Rule and Notation

Let F be a formula containing the variable p. The notation $F[p \leftarrow G]$ (or $F[G/p]$) means: Replace every occurrence of p in F with the formula G.
Important Rule: The replacement must be simultaneous. You cannot change one instance of p while leaving another unchanged.

Example: Given the formula $F = (p \Rightarrow q) \wedge p$. Suppose we substitute p with $(r \vee s)$:
Locate all p's: $(\underline{p} \Rightarrow q) \wedge \underline{p}$
Replace them: $((r \vee s) \Rightarrow q) \wedge (r \vee s)$

Enriched Explanation (The Substitution Theorem): Why is this essential? If we prove that a formula like $(p \vee \neg p)$ is always true (a tautology), the Substitution Theorem tells us that replacing p with any complex formula, no matter how large, will also result in a tautology.
Example: If $(p \vee \neg p)$ is True, then $((A \wedge B) \vee \neg(A \wedge B))$ is also True by substitution. This is a core technique in Logic Programming and automated theorem proving.

4.6 Syntactic and Semantic Validity

As we conclude this session, it is vital to distinguish between Syntactic Validity (Well-Formedness), which means that the formula is grammatically correct. Semantic Validity (Tautology) where the formula is True under every possible interpretation of its variables. These two notions must never be confused: a formula can be well-written but still be False.

4.7 Summary

Session IV established the formal grammar of logic. We learned that propositional formulas are built recursively from variables and constants and that they can be manipulated via substitution to prove complex properties. This symbolic manipulation is the foundation for all subsequent chapters on truth tables and logical proofs.

Go to Exercises

5 Session V: Logical Formulas and Syntax Trees

5.1 Introduction and Motivation

In Session V, we conclude our study of the syntax of Propositional Logic by transitioning from linear strings of symbols to structural representations. While we write formulas in a straight line (e.g., $p \wedge q \Rightarrow r$), computers and logic systems perceive them as a hierarchy.

Understanding logic as a tree is a vital skill for CS students, as this is exactly how compilers and parsers see and evaluate code. Syntax trees are standard tools in compiler design, symbolic computation, and automated theorem proving.

5.2 Structural Representation: Why Trees?

A Syntax Tree (also known as a Construction Tree or Parse Tree) is a graphical representation of the internal structure of a formula. It reflects the hierarchical application of logical connectives and eliminates the need for parentheses.

Components of a Logic Tree:
Leaves (Terminal Nodes): These are the base of the tree and are always Propositional Variables (p, q, r).
Internal Nodes: These represent the Logical Connectives ($\neg, \wedge, \vee, \Rightarrow, \Leftrightarrow$).
The Root Node: The connective at the very top of the tree is the Main Connective. This represents the last operation that would be performed if you were calculating the truth value of the formula.

5.3 Construction of a Syntax Tree

To draw a tree correctly, one must follow the Precedence Rules established in Session IV. The tree is built top-down from the main connective.

Steps for Construction:

1. Identify the Main Connective: Find the operator that has the widest scope (the one outside of all parentheses or the one with the lowest priority). This becomes the Root.
2. Split the Formula: Divide the formula into its subformulas.
   - Binary operators ($\wedge, \vee, \dots$) produce a left branch and a right branch.
   - Unary operators ($\neg$) produce a single downward branch.
3. Recursive Decomposition: Repeat the process for each sub-part until every branch ends in a propositional variable (a leaf).

Construction Example: Let us draw the syntax tree for the formula: $\phi = \neg(p \vee q) \Leftrightarrow (p \wedge r)$.
Main Connective: The $\Leftrightarrow$ is the root, as it connects the two largest logical blocks.
Left Branch: The subformula is $\neg(p \vee q)$. The next node is $\neg$, which leads to the node $\vee$, which then splits into leaves p and q.
Right Branch: The subformula is $(p \wedge r)$. The next node is $\wedge$, splitting into leaves p and r.

5.4 Subformulas

A subformula of a formula ϕ is any part of ϕ that is itself a Well-Formed Formula (WFF), including ϕ itself.

Identifying Subformulas via the Tree: Every node in a syntax tree serves as the root of a specific subformula. Visualizing the tree makes it easy to list the entire Set of Subformulas.
Example: If $\phi = (\neg p \wedge q)$, the subformulas are:

p (Atomic subformula)
q (Atomic subformula)
$\neg p$ (Compound subformula)
$(\neg p \wedge q)$ (The formula itself)

Importance of Subformulas: Subformulas are the essential building blocks of logical expressions. They play a central role in semantic evaluation, where truth values are computed step by step from the atomic components (leaves) to the complete formula (root). Subformulas are also fundamental in normal form transformations, allowing logical expressions to be rewritten into standard representations such as Conjunctive Normal Form (CNF) or Disjunctive Normal Form (DNF). In addition, proof procedures such as Resolution and Natural Deduction rely heavily on the systematic decomposition of formulas into their subformulas.

5.5 Advanced Insight: Structural Induction

Because formulas are built recursively like trees, we can use a powerful mathematical technique called Structural Induction to prove properties about all possible formulas.

> The Logic: If a property holds for the leaves (atoms) and we can prove that the property is preserved when we combine subformulas using connectives, then the property must hold for all formulas in the language.

This is the foundation of Formal Verification, proving that software code or a hardware circuit is correct for every possible input without testing every single case.

5.6 Summary

Syntax trees provide a powerful way to understand and work with logical formulas by making their structure explicit. Through visualization, they reveal the hierarchical organization that is hidden in linear string representations. In a syntax tree, internal nodes correspond to logical connectives, while the leaves represent variables. This hierarchical structure precisely determines the scope of every variable and operator, which in turn simplifies formula manipulation and analysis by algorithms.

Go to Exercises

6 Session VI: Semantics - Interpretation and Truth Tables

6.1 Introduction: From Form to Meaning

In Session VI, we leave the Form (Syntax) and enter the Meaning (Semantics). For a computer scientist, this is the equivalent of moving from writing code to actually executing it to observe the output. While syntax tells us if a formula is well-formed, semantics tells us whether it is True or False.

6.2 The Concept of Interpretation (Valuation)

In syntax, p is merely a symbol. In semantics, p represents a state of reality.

Definition of an Interpretation
An Interpretation (also called a Valuation) is a function I (or v) that maps the set of propositional variables $\mathcal{P}$ to truth values $\{T, F\}$ (or $\{1, 0\}$).
Function Mapping: $I : \mathcal{P} \to \{T, F\}$.
Combinatorics: If a formula has n distinct variables, there are 2^n possible interpretations.
Example: For the set $\{p, q\}$, one possible interpretation is $I(p) = T, I(q) = F$.

Truth Value of a Formula: The goal of semantics is to determine the truth value of a complex formula ϕ under a specific interpretation I, denoted as $I(\phi)$. This is defined recursively based on the syntax tree:

$$\begin{aligned}
&I(\neg\phi) = T \text{ iff } I(\phi) = F.\\
&I(\phi \wedge \psi) = T \text{ iff both } I(\phi) = T \text{ and } I(\psi) = T.\\
&I(\phi \vee \psi) = T \text{ iff at least one is } T.\\
&I(\phi \to \psi) = F \text{ iff } I(\phi) = T \text{ and } I(\psi) = F.\\
&I(\phi \leftrightarrow \psi) = T \text{ iff both have the same truth value.}
\end{aligned}$$

6.3 Standard Truth Tables of Connectives

To calculate $I(\phi)$, we must master the behavior of each connective.

Unary and Binary Operators
Negation ($\neg$): The value is flipped (unary operator).
Conjunction ($\wedge$): Acts like a series circuit; it is True only if both components are 1.
Disjunction ($\vee$): Acts like a parallel circuit; it is True if at least one path is 1.
Implication ($\to$): Crucial Remark: Only False when the premise is True but the conclusion is False ($1 \to 0$). If the premise is False, the result is True regardless of the conclusion (this is known as Vacuous Truth).
Biconditional ($\leftrightarrow$): True if values match ($1 \leftrightarrow 1$ or $0 \leftrightarrow 0$).

6.4 Constructing Truth Tables for Formulas

A truth table lists every possible interpretation of the variables and the resulting value of the formula for each.

Systematic Method: Three steps are needed to construct a Truth Table.

1. Count Variables (n): Create 2^n rows.
2. List Subformulas: Create columns for variables, then for subformulas in order of their construction (leaves to root).
3. Fill Columns: Calculate values step-by-step using the rules in Section 3.

Complex Example: $\phi = (p \wedge q) \to (r \vee \neg p)$ With 3 variables (p, q, r), we have $2^3 = 8$ rows. Below is an excerpt showing the step-by-step evaluation.

p	q	r	$p \wedge q$	$\neg p$	$r \vee \neg p$	$(p \wedge q) \to (r \vee \neg p)$
1	1	0	1	0	0	**0**
1	1	1	1	0	1	**1**
0	1	0	0	1	1	**1**

Enriched Context (Logic and Circuits): In Computer Science, these truth tables are the blueprints for Logic Gates. An AND Gate implements $p \wedge q$, an OR Gate implements $p \vee q$, and a NOT Gate (Inverter) implements $\neg p$. When we write a complex "if" statement in a high-level language, the compiler uses these semantic rules to generate the machine-level jumps and comparisons.

Why Semantics Matters: Truth tables are more than simple exercises; they play a crucial role in understanding and applying logic. They allow us to determine the *validity* of a formula by checking whether it is always true, that is, a tautology. Truth tables also make it possible to verify whether two syntactically different formulas are logically *equivalent* and thus share the same meaning. Additionally, they are essential for system *verification*, as they ensure that for every possible interpretation or state of a system, the resulting output or conclusion remains correct and safe.

6.5 Summary

Session VI provided the tools to evaluate the Soul (meaning) of our logical Body (syntax). By enumerating all possible worlds (interpretations), we can prove properties of formulas with mathematical certainty.

Go to Exercises

7 Session VII: Classification of Formulas and Semantic Equivalence

7.1 Introduction and Motivation

Once we have mastered the computation of truth values through interpretations and truth tables, we can move beyond analyzing individual cases to classifying formulas according to their universal behavior. This shift allows us to distinguish between logical laws, which are statements that are always true, and logical flaws, which are statements that are never true. This distinction forms a fundamental basis for logical reasoning, as it helps identify valid structures of thought. It is also essential in program verification, where we aim to prove that code behaves correctly under all possible inputs. In addition, it plays a key role in digital circuit design by enabling the simplification of logic gates to reduce hardware costs, and it underpins automated theorem proving, which allows artificial intelligence systems to derive new truths from established facts.

7.2 Classification by Truth Table

After completing a truth table, we examine the final column (the result of the main connective). Based on the distribution of 1s and 0s, every formula falls into one of three categories.

a. Tautology (Logical Law)

A Tautology is a True formula ($T/1$) under every possible interpretation. It represents a universal truth.
Notation: $\models \phi$
Definition: $I(\phi) = T$ for all interpretations I.
Classic Example: $p \vee \neg p$ (The Law of Excluded Middle). A statement is either true, or its negation is true; there is no third option.
Logical Meaning: "If both p and q are true, then p is true" ($(p \wedge q) \rightarrow p$) is always valid.

b. Antilogy (Contradiction)

An Antilogy (or Contradiction) is a formula that is False ($F/0$) under every possible interpretation. It represents a logical impossibility.
Definition: $I(\phi) = F$ for all interpretations I.
Classic Example: $p \wedge \neg p$. A statement and its negation cannot both be true simultaneously.
In CS Context: An antilogy represents a Bug or an unreachable, impossible state in a software system.

c. Contingency

A Contingency is a formula that is True for some interpretations and False for others. Most statements in daily programming and natural language are contingencies.
Example: $p \Rightarrow q$. This is false if $p = 1, q = 0$, but true in the other three cases. It is neither a tautology nor an antilogy.

7.3 Semantic Equivalence

Two formulas ϕ and ψ are Semantically Equivalent ($\equiv$) if they share the exact same truth value under every possible interpretation. In other words, their final truth table columns are identical.

The Relationship with the Biconditional:
Two formulas ϕ and ψ are equivalent ($\phi \equiv \psi$) if and only if the formula $\phi \Leftrightarrow \psi$ is a Tautology.

Key Logical Equivalences (Laws of Logic): Mastering these laws allows computer scientists to simplify complex logic and transform formulas into Normal Forms:

De Morgan's Laws: $\neg(p \wedge q) \equiv \neg p \vee \neg q$, $\neg(p \vee q) \equiv \neg p \wedge \neg q$
Double Negation: $\neg\neg p \equiv p$
Implication Elimination: $p \rightarrow q \equiv \neg p \vee q$ (Essential for converting logic to hardware-friendly formats).
Commutativity: $p \wedge q \equiv q \wedge p$ and $p \vee q \equiv q \vee p$.
Distributivity: $p \wedge (q \vee r) \equiv (p \wedge q) \vee (p \wedge r)$.
Biconditional: $(p \leftrightarrow q) \equiv (p \rightarrow q) \wedge (q \rightarrow p)$.

Why This Matters for Computing: In Software Engineering, we frequently simplify complex conditional logic to improve readability and performance.
Example of Code Optimization:

Original Code: `if (!(!A && !B))`
Logical Breakdown: $\neg(\neg A \wedge \neg B)$
Applying De Morgan's Law: $\neg(\neg A) \vee \neg(\neg B)$
Applying Double Negation: $A \vee B$
Simplified Code: `if (A || B)`

The simplified version is easier to read, faster to execute, and significantly less prone to human error.

Preview (Validity and Satisfiability): While we have classified formulas by their total behavior, we will soon look at partial behavior.
A formula is Valid $\Leftrightarrow$ It is a Tautology.
A formula is Satisfiable $\Leftrightarrow$ It is True under at least one interpretation.
An Antilogy is Unsatisfiable (True under zero interpretations).

7.4 Summary

Tautologies express logical laws, while antilogies represent logical impossibilities. Semantic equivalence allows us to replace complex formulas with simpler ones without altering their meaning. Truth tables remain the primary tool for verifying these classifications and equivalences.

Go to Exercises

8 Session VIII: Normal Forms, Satisfaction, and Validity

8.1 Introduction and Motivation

In Session VIII, we move toward standardizing logic. In Computer Science, algorithms (such as SAT solvers and automated theorem provers) require data to be in a specific format to process it efficiently. These formats are known as Normal Forms.

Just as every fraction can be simplified (e.g., 2/4 to 1/2), every logical formula can be transformed into a Normal Form that is syntactically standard but semantically equivalent to the original.

8.2 Building Blocks: Literals and Clauses

Before defining normal forms, we must define the units used to build them:
Literal: A propositional variable (p) or its negation ($\neg p$).
Clause: A disjunction (OR) of literals. Example: $(p \vee \neg q \vee r)$.
Dual Clause (Term): A conjunction (AND) of literals. Example: $(p \wedge \neg q \wedge r)$.

8.3 Disjunctive Normal Form (DNF)

A formula is in DNF if it is a sum of products, specifically, a disjunction of conjunctions.
Structure: $(C_1 \wedge C_2 \dots) \vee (C_3 \wedge C_4 \dots) \vee \dots$
Usage: DNF is highly useful for determining satisfiability. If at least one conjunction (one bracket) is not a contradiction, the whole formula is satisfiable.
Example: $(p \wedge \neg q) \vee (\neg p \wedge r)$ is in DNF.

8.4 Conjunctive Normal Form (CNF)

A formula is in CNF if it is a product of sums, specifically, a conjunction of clauses.
Structure: $(L_1 \vee L_2 \dots) \wedge (L_3 \vee L_4 \dots) \wedge \dots$
Usage: CNF is the standard for automated theorem proving and the resolution method. Most modern logic software (SAT Solvers) requires input in CNF.
Example: $(p \vee q) \wedge (\neg p \vee r)$ is in CNF.

8.5 Formalizing Satisfaction, Validity, and Models

These terms provide the mathematical framework for our semantic analysis.

Satisfaction and Models
Satisfaction: An interpretation v satisfies a formula F if $v(F) = 1$.
Model: In this case, v is called a Model of F.
Notation: $v \models F$.

Satisfiability vs. Validity: A formula is said to be *satisfiable* if there exists at least one interpretation (or model) under which the formula evaluates to true. Both tautologies, which are true under all interpretations, and contingencies, which are true under some interpretations and false under others, are satisfiable. In contrast, a formula is *unsatisfiable* if there is no interpretation that makes it true; such a formula is also known as an *antilogy*.

A formula is *valid* if it is true under every possible interpretation, in which case it is a tautology. An important logical law relates validity and satisfiability: a formula F is valid if and only if its negation $\neg F$ is unsatisfiable.

8.6 Constructing Normal Forms from Truth Tables

We can derive Full or Canonical Normal Forms by looking at the results of a truth table.

How to Find the Disjunctive Normal Form (DNF): The DNF of a propositional formula can be constructed directly from its truth table. The idea is to identify all cases in which the formula evaluates to **1** (true) and translate each such case into a logical expression.

1. Examine the rows of the truth table where the result is **1**.

2. For each such row, construct a conjunction (AND) of all the variables. This conjunction represents a specific assignment of truth values that makes the formula true.
3. If a variable has the value 0 (false) in that row, replace it with its negation; if it has the value 1 (true), keep it unchanged.
4. Finally, combine all the resulting conjunctions using the disjunction operator $\vee$. The resulting formula is in Disjunctive Normal Form.

How to Find the Conjunctive Normal Form (CNF): The Conjunctive Normal Form of a propositional formula can also be constructed directly from its truth table. In this case, the focus is on the rows where the formula evaluates to **0** (false). Each such row describes a situation that must be excluded by the final formula.

1. Examine the rows of the truth table where the result is **0**.
2. For each such row, construct a disjunction (OR) of all the variables. This disjunction represents a condition under which the formula would be false.
3. If a variable has the value **1** (true) in that row, replace it with its negation; if it has the value **0** (false), keep it unchanged. This rule is the inverse of the one used for constructing the DNF.
4. Finally, combine all the resulting disjunctions using the conjunction operator $\wedge$. The resulting formula is in Conjunctive Normal Form.

This makes the DNF/CNF duality very clear:

DNF $\to$ rows where the result is **true**, joined by $\vee$ (OR).
CNF $\to$ rows where the result is **false**, joined by $\wedge$ (AND).

8.7 Algebraic Transformation to CNF

Instead of truth tables, we can use logical laws to transform formulas.
Elimination: Remove $\to$ and $\leftrightarrow$ (e.g., $p \to q \equiv \neg p \vee q$).
Negation: Push $\neg$ inward using De Morgan's Laws and remove double negations.
Distribution: Use $p \vee (q \wedge r) \equiv (p \vee q) \wedge (p \vee r)$ to ensure the $\wedge$ is at the top level.

8.8 Summary

Session VIII moved us from raw formulas to standardized structures. We defined CNF (conjunction of clauses) and DNF (disjunction of terms). We learned that CNF is crucial for automated reasoning and that a formula's truth can be categorized as Valid, Satisfiable, or Unsatisfiable. These tools prepare us for the Resolution Method in the next session.

Go to Exercises

9 Session IX: The Resolution Method - Refutation and Clausal Form

9.1 Introduction and Motivation

In Session IX, we move away from the use of truth tables, whose size grows exponentially as 2^n and quickly becomes inefficient for large-scale logical problems. Instead, we introduce the Resolution Method, which forms the core logical mechanism behind modern automated reasoning tools such as Prover9 and Z3, as well as logic programming languages like Prolog.

The focus of this session is on three fundamental notions in propositional logic. The first is *satisfiability*, which asks whether a given formula can be true under at least one interpretation. The second is *validity*, which concerns whether a formula is true under all possible interpretations. Finally, *unsatisfiability* addresses whether a given set of formulas is inconsistent and admits no interpretation that makes all formulas true simultaneously.

9.2 Formal Satisfaction and Models

To perform automated reasoning, we must formalize what it means for a formula to be true in a given context.

Satisfaction of a Formula and Sets

Single Formula: An interpretation I satisfies a formula ϕ (written $I \models \phi$) if $I(\phi) = T$.

Set of Formulas: An interpretation I satisfies a set $\Gamma = \{\phi_1, \phi_2, \ldots, \phi_n\}$ if it satisfies every formula in that set simultaneously. We write this as $I \models \Gamma$.

Satisfiable Set: A set is satisfiable if there exists at least one interpretation that makes every formula in the set true. If no such interpretation exists, the set is inconsistent.

Summary of Properties:

Property	Meaning
Valid	True under all interpretations (Tautology).
Satisfiable	True under at least one interpretation.
Unsatisfiable	True under none (Antilogy / Contradiction).

9.3 The Concept of Refutation (Proof by Contradiction)

Resolution is a Refutation-based method. Instead of proving that a conclusion D follows from premises $\{A, B, C\}$ directly, we take the opposite approach:

The Setup: Assume the premises $\{A, B, C\}$ are true and the conclusion is false ($\neg D$).

The Goal: Show that the set $\{A, B, C, \neg D\}$ is Unsatisfiable.

The Logic: If it is impossible for the negation of the conclusion to be true alongside the premises, then the original conclusion must be true.

9.4 Clausal Form: The Language of Resolution

The Resolution algorithm cannot process standard formulas with implications or nested parentheses. It requires a specific format called a Clause Set.

Definitions

Literal: A variable p (positive) or its negation $\neg p$ (negative).

Clause: A disjunction (OR) of literals. In set notation, $\{p, \neg q, r\}$ represents $(p \vee \neg q \vee r)$.

Empty Clause ($\square$ or $\bot$): A clause with no literals. It represents False or a Contradiction.

Clause Set (Σ): A conjunction (AND) of clauses. This is essentially a formula in Conjunctive Normal Form (CNF).

Transforming a Formula into a Clause Set: To prepare a formula for resolution, follow these four mechanical steps:

1. Eliminate Implications: Replace $(A \to B)$ with $(\neg A \vee B)$.
2. Reduce Negation Scope: Move $\neg$ inward using De Morgan's Laws until it only touches variables ($\neg\neg p \equiv p$).
3. Standardize to CNF: Use Distributive Laws ($p \vee (q \wedge r) \equiv (p \vee q) \wedge (p \vee r)$) to ensure the structure is

a conjunction of disjunctions.
4. Extract Clauses: Write each bracketed group as a set of literals within a larger set.
Example: Transform $\neg(p \to q) \lor r$ into a Clause Set.

Step 1 (Implication): $\neg(\neg p \lor q) \lor r$
Step 2 (De Morgan): $(p \land \neg q) \lor r$
Step 3 (Distributivity): $(p \lor r) \land (\neg q \lor r)$
Step 4 (Clause Set): $\{\{p, r\}, \{\neg q, r\}\}$

9.5 Why Clausal Form is Crucial

Automated solvers use clausal form because it simplifies the search for a contradiction. Resolution only needs to look at pairs of clauses to see if they contain opposing literals (e.g., p and $\neg p$). If we can eventually derive the Empty Clause ($\square$), we have found a mathematical contradiction, proving the original argument is valid.

9.6 Summary

Session IX moved us from truth tables to the syntax of proof. We formalized satisfaction for sets, defined the refutation strategy, and learned how to translate complex formulas into the Clausal Form required for the Resolution Rule.

Go to Exercises

10 Session X: The Propositional Resolution Method

10.1 Introduction: The Climax of Propositional Logic

In Session X, we reach the climax of Chapter 2. After mastering clausal form, we introduce a single powerful rule that enables a computer to determine whether a conclusion follows as a logical necessity. This rule, known as *resolution*, is a purely syntactic and mechanical inference rule, making it particularly well suited for automation.

Resolution forms the theoretical foundation of several important technologies in computer science. It lies at the heart of SAT solvers, which are widely used in hardware verification and complex scheduling problems. It is also the basis of automated theorem provers, which are capable of discovering mathematical proofs, and of logic programming, providing the inference engine behind languages such as Prolog.

10.2 The Propositional Resolution Rule

The Resolution rule produces a new clause (the Resolvent) from two existing clauses that contain complementary literals.

Complementary Literals: Two literals are complementary if one is the negation of the other (e.g., p and $\neg p$). These represent a clash in the logical state.

The Formal Rule: Let C_1 and C_2 be two clauses. If C_1 contains a literal l and C_2 contains its negation $\neg l$:

$$\frac{l \lor A \quad \neg l \lor B}{A \lor B}$$

Where A and B are the remaining literals in each clause. We cancel out the complementary pair and merge what remains.

Intuitive Meaning: Consider a literal l and two clauses that involve l.
If l is True, then in order for the second clause to be satisfied, B must also be True.
Conversely, if l is False, then in order for the first clause to be satisfied, A must be True.
Since l can only take one of these two truth values (True or False), at least one of A or B is guaranteed to be true in all cases.
Therefore, the disjunction $A \lor B$ is always true, illustrating how the truth of $A \lor B$ is logically ensured by considering the possible truth values of l.

Simple Examples

Basic Merge: From $\{p, q\}$ and $\{\neg p, r\}$, resolving on p yields $\{q, r\}$.
Modus Ponens: From $\{p\}$ and $\{\neg p, q\}$ (which is $p \Rightarrow q$), resolving on p yields $\{q\}$.
Contradiction: From $\{p\}$ and $\{\neg p\}$, resolving yields the Empty Clause ($\square$ or $\bot$).

10.3 The Resolution Refutation Algorithm

To prove that a set of premises logically implies a conclusion ($F \models G$), we use the Refutation Procedure:

1. Negate the Goal: Take the conclusion you want to prove and negate it ($\neg G$).
2. Convert to Clauses: Transform all premises and the negated conclusion into a set of clauses S (using the CNF steps from Session IX).
3. Iterative Resolution:

   a. Pick two clauses with complementary literals.
   b. Compute their resolvent.
   c. Add the resolvent back into the set S.

4. Termination:

   a. Success: If you derive the Empty Clause ($\square$), a contradiction is found. The original conclusion G is Valid.
   b. Failure: If no more new resolvents can be created and $\square$ was never found, the set is satisfiable. The conclusion G does not follow from the premises.

Full Example: Proving Validity
Problem: Prove that $(p \wedge q) \rightarrow p$ is valid.
Step 1 (Negate): $\neg((p \wedge q) \rightarrow p)$
Step 2 (Transform): $\neg(\neg(p \wedge q) \vee p) \equiv (p \wedge q) \wedge \neg p$
Step 3 (Clause Set): $\{\{p\}, \{q\}, \{\neg p\}\}$
Step 4 (Resolve): Pick $\{p\}$ and $\{\neg p\}$. Resolvent: $\Box$.
Step 5 (Conclusion): Contradiction found ($\Box$). The original formula is Valid.

Why We Trust Resolution: Resolution is a reliable reasoning method because it satisfies three fundamental theoretical properties. First, it is *sound*, meaning that it never derives false conclusions: if resolution derives the empty clause $\Box$, then the original set of clauses is guaranteed to be unsatisfiable. Second, resolution is *complete*, which ensures that if a set of clauses is unsatisfiable, the resolution process will eventually derive the empty clause.

Finally, resolution is *decidable* in the context of propositional logic. Since the number of possible distinct clauses over a finite set of propositional variables is itself finite, the resolution algorithm is guaranteed to terminate and will not run indefinitely.

10.4 Summary

In Session X, we synthesized everything from Chapter 2. We used syntax (Clausal Form) and semantics (Satisfiability) to create a mechanical proof system. We now have a logic engine that can evaluate the validity of any argument by searching for a contradiction.

Go to Exercises

11 Session XI: Predicate Logic - The Language of Objects

11.1 Introduction: Moving Beyond Propositional Limits

In Session XI, we transition from Propositional Logic to Predicate Logic (also known as First-Order Logic or FOL). While Propositional Logic is powerful, it is blind to the internal structure of sentences.

The Problem:
In Propositional Logic: "Socrates is a man" $= P$, "Plato is a man" $= Q$
There is no mathematical way to show that P and Q share the property of being a man. Propositional logic can say "It is raining," but it cannot easily say "Everyone who is outside is getting wet." Predicate Logic allows us to look inside a proposition to describe objects, their properties, and their quantities.

11.2 The Syntax of Terms (The Nouns)

In FOL, Terms are expressions that represent objects in our Domain of Discourse. Think of these as the nouns of our logical language.

11.3 Terms

We have three terms.
Constants: Symbols that represent a specific, fixed object. Examples: s (Socrates), 5 (the number 5), a (Algiers).
Variables: Symbols that act as placeholders for objects. They do not point to one specific thing until we assign them a value or quantify them. Examples: x, y, z.
Functions: Functions take one or more terms and return a single object.

Example 1: $father(x)$ represents the father of x.
Example 2: $plus(x, y)$ represents the result of adding x and y.

Note: A function is not a proposition. "The father of Ali" is an object, not a True/False statement.

11.4 Predicates (The Verbs and Adjectives)

A Predicate represents a property of an object or a relationship between objects. Applying a predicate to a term creates a Formula (something that can be True or False).
Unary Predicate (Properties): $M(x)$ could mean "x is a Man."
Binary Predicate (Relations): $L(x, y)$ could mean "x loves y."
n-ary Predicate: $B(x, y, z)$ could mean "x sits between y and z."

11.5 Quantifiers (The Quantity)

Quantifiers allow us to talk about groups of objects without naming every individual object in the universe.

The Universal Quantifier
Symbol: $\forall$ (Read as: For all, For every, or Each).
Meaning: The property holds for every object in the domain.
Example: $\forall x M(x)$ means "Everything in the domain is a man."

The Existential Quantifier
Symbol: $\exists$ (Read as: There exists or: For at least one).
Meaning: The property holds for at least one object in the domain.
Example: $\exists x M(x)$ means "There is at least one man."

11.6 Formulas

Having introduced *terms* (which denote objects) and *predicates* (which express properties and relations among objects), we now focus on *formulas*, which are the central syntactic expressions of predicate logic. Formulas represent complete statements that can later be evaluated as true or false once a semantic interpretation is provided.

Atomic Formulas. The simplest formulas are called *atomic formulas*. An atomic formula is obtained by applying a predicate symbol to the appropriate number of terms. If P is an n-ary predicate symbol and $t_1, \ldots, t_n$ are terms, then

$$P(t_1, \ldots, t_n)$$

is an atomic formula.
Examples include:

$$Human(x), \quad Loves(x, y), \quad GreaterThan(s(0), y)$$

Complex Formulas: Using the formation rules introduced in the previous subsection, atomic formulas can be combined using logical connectives. If φ and ψ are formulas, then expressions such as

$$\neg\varphi, \quad (\varphi \wedge \psi), \quad (\varphi \vee \psi), \quad (\varphi \to \psi)$$

are also formulas.

This shows that predicate logic extends propositional logic by allowing formulas to contain internal structure while preserving the same logical operators.

Quantified Formulas: Predicate logic further allows the use of quantifiers to express statements about collections of objects. If φ is a formula and x is a variable, then:

$$\forall x\, \varphi \quad \text{and} \quad \exists x\, \varphi$$

are formulas, representing universal and existential statements, respectively.

Remark: In this session, formulas are treated purely as *syntactic objects*. Their meaning, the role of variable scope, and the conditions under which a formula is true or false will be introduced in the next session on semantics.

Summary of Propositional and Predicate Logic

Feature	Propositional Logic	Predicate Logic (FOL)
Basic Unit	Complete Sentence (P)	Objects (x), Properties ($P(x)$)
Granularity	Coarse (atomic)	Fine (internal structure)
Generalization	Impossible	Possible via Quantifiers ($\forall, \exists$)
Complexity	Lower (Decidable)	Higher (Semi-decidable)

11.7 Summary

Session XI has equipped us with a much more expressive language. We can now describe complex worlds, mathematical structures, and database relations by identifying the objects involved, the relationships between them, and the extent to which those relationships exist.

Go to Exercises

12 Session XII: Predicate Logic - Scope, Variables, and Semantics

12.1 Introduction and Motivation

While propositional logic allows us to reason about simple atoms of information, it remains blind to the relationships between specific objects. In this final session, we complete the study of the syntax of predicate logic (first-order logic) by examining how variables are controlled by quantifiers, and we then turn to semantics, which explains how the truth of a statement is determined within a given world or structure.

Predicate logic serves as a fundamental framework for many core areas of computer science. It underlies database systems, where SQL queries can be viewed as expressions of first-order logic. It is also central to artificial intelligence, particularly in knowledge representation and automated reasoning, and to formal verification, where it is used to prove that programs behave correctly for all possible inputs.

12.2 Syntax: Scope and Variables

In Predicate Logic, the life of a variable is managed by its Quantifier. Understanding which variable belongs to which quantifier is essential for correct evaluation.

Scope of a Quantifier

The Scope is the specific part of the formula to which a quantifier applies. In visual terms, it is the reach of the quantifier.

Example 1: $\forall x(P(x) \Rightarrow Q(x))$. The scope of $\forall x$ is $(P(x) \Rightarrow Q(x))$. Both x's are governed by the quantifier.

Example 2: $(\forall x P(x)) \Rightarrow Q(x)$. The scope of $\forall x$ is only $P(x)$. The x in $Q(x)$ is outside and unrelated to the first x.

Free and Bound Variables

Bound Variable: A variable is bound if it occurs within the scope of a quantifier (e.g., x in $\exists x P(x)$).
Free Variable: A variable is free if it is not governed by any quantifier. It acts as a dangling placeholder.
Sentence (Closed Formula): A formula with no free variables. In logic, only sentences can be definitively True or False once an interpretation is provided.

12.3 Semantics: Structures and Interpretation

In propositional logic, an interpretation is simply an assignment of truth values (0 or 1) to propositional variables. In predicate logic, however, an interpretation (also called a structure $\mathcal{M}$) is a much richer mathematical object.

The Components of a Structure

A structure ($\mathcal{M}$) defines the World we are talking about. It consists of:
The Domain (D): A non-empty set of all objects in our universe (e.g., all Integers $\mathbb{N}$, all employees in a company).
Assignments for Constants: Mapping symbol names (like a) to specific objects in D.
Assignments for Predicates: Defining which objects or pairs of objects satisfy a property (e.g., mapping $Prime(x)$ to the actual set of prime numbers).
Assignments for Functions: Defining the rules for mapping inputs to outputs (e.g., mapping $add(x, y)$ to the mathematical addition).

12.4 Satisfaction and the Tarski Definition of Truth

We say a structure $\mathcal{M}$ satisfies a formula ϕ (written $\mathcal{M} \models \phi$) if the formula evaluates to True within that specific domain.

Example: The Power of the Domain
Consider the formula: $\exists\, x\ (x \times x = x)$.
Case 1: If $D = \{2, 3, 4\}$, the formula is False ($\mathcal{M} \not\models \phi$) because no number in the set multiplied by itself equals itself.
Case 2: If $D = \{0, 1, 2\}$, the formula is True ($\mathcal{M} \models \phi$) because $1 \times 1 = 1$.

Simplified Truth Conditions
$\mathcal{M} \models \forall x P(x)$ iff $P(x)$ is true for every single element in D.
$\mathcal{M} \models \exists x P(x)$ iff $P(x)$ is true for at least one element in D.

12.5 Summary Table: Propositional and Predicate Logic

Aspect	Propositional Logic	Predicate Logic (FOL)
Focus	Statements as a whole	Objects and their relations
Vocabulary	P, Q, R	Constants, Variables, Functions
Quantifiers	None	Universal ($\forall$) and Existential ($\exists$)
Interpretation	Truth assignment (v)	Structure ($\mathcal{M}$) with Domain (D)

12.6 Final Wrap-up

Session XII marks the end of our journey through the fundamentals of logic. We have moved from simple truth tables to complex structures capable of describing the entire mathematical universe. By distinguishing between Bound and Free variables and defining Satisfaction through domains, we have built the formal tools necessary for modern computer science.

Go to Exercises

General Conclusion

Across these twelve sessions, we have traversed the landscape of formal logic, moving from the microscopic level of symbols and connectives to the macroscopic level of mathematical structures and domains. Therefore, by the end of this course, students will have acquired a solid understanding of the principles of mathematical logic, including the formal structure, interpretation, and manipulation of logical expressions in both propositional and predicate logic. They will be able to analyze logical statements, determine their validity, transform formulas into equivalent normal forms, and apply resolution-based reasoning techniques.

We have seen that logic is not merely a theoretical exercise; it is a practical tool for standardization, where it uses normal forms (CNF/DNF) to prepare data for algorithmic processing. For optimization, it uses logical equivalences and De Morgan's Laws to simplify complex code and digital circuits. For verification, by using the resolution method to prove the validity of arguments and the safety of systems. And for modeling, by using predicate logic to define the properties and relationships of objects within a computational domain.

The progressive organization of the course content, combined with the close alignment between lectures and tutorials, ensures that theoretical knowledge is consistently reinforced through practical application. This approach enables students to develop precision, rigor, and logical clarity in their reasoning processes.

Finally, this course lays a strong theoretical foundation for further studies in areas such as algorithms, formal verification, artificial intelligence, and information security, where logical reasoning and formal methods play a central role.

End of Course Modules

Appendix: De Morgan's Laws Summary

In logic and computer programming, De Morgan's Laws allow us to distribute or factor out a negation over conjunctions ($\wedge$) and disjunctions ($\vee$).

12.7 The Laws

Negation of a Conjunction:

$$\neg(P \wedge Q) \equiv \neg P \vee \neg Q$$

Not (A and B) is the same as Not A or Not B.

Negation of a Disjunction:

$$\neg(P \vee Q) \equiv \neg P \wedge \neg Q$$

Not (A or B) is the same as Not A and Not B.

Application in Programming

When you see a complex Not condition in code, like `!(A || B)`, you can simplify it by flipping the operators and the connectives:
Original: `if (!(age < 18 || has_permission))`
Simplified: `if (age >= 18 && !has_permission)`

Appendix: Fundamental Laws of Propositional Logic

This table summarizes the logical equivalences used to simplify complex formulas and prove semantic identities.

Law Name	Conjunction ($\wedge$)	Disjunction ($\vee$)
Identity	$P \wedge \top \equiv P$	$P \vee \bot \equiv P$
Dominance	$P \wedge \bot \equiv \bot$	$P \vee \top \equiv \top$
Idempotent	$P \wedge P \equiv P$	$P \vee P \equiv P$
Double Negation	$\neg(\neg P) \equiv P$	
Commutative	$P \wedge Q \equiv Q \wedge P$	$P \vee Q \equiv Q \vee P$
Associative	$(P \wedge Q) \wedge R \equiv P \wedge (Q \wedge R)$	$(P \vee Q) \vee R \equiv P \vee (Q \vee R)$
Distributive	$P \wedge (Q \vee R) \equiv (P \wedge Q) \vee (P \wedge R)$	$P \vee (Q \wedge R) \equiv (P \vee Q) \wedge (P \vee R)$
De Morgan's	$\neg(P \wedge Q) \equiv \neg P \vee \neg Q$	$\neg(P \vee Q) \equiv \neg P \wedge \neg Q$
Absorption	$P \wedge (P \vee Q) \equiv P$	$P \vee (P \wedge Q) \equiv P$
Negation	$P \wedge \neg P \equiv \bot$	$P \vee \neg P \equiv \top$

Implication and Biconditional Identities

These are essential for transforming formulas into different sets of connectives.
Implication Identity: $P \Rightarrow Q \equiv \neg P \vee Q$
Contrapositive: $P \Rightarrow Q \equiv \neg Q \Rightarrow \neg P$
Biconditional Identity: $P \Leftrightarrow Q \equiv (P \Rightarrow Q) \wedge (Q \Rightarrow P)$

References

1. Avigad, Jeremy. Mathematical Logic and Computation. Cambridge University Press, 2023
2. Ben-Ari, Mordechai. Mathematical Logic for Computer Science. 3rd Edition, Springer, 2012
3. Eric Lehman, F Thomson Leighton, Albert R Meyer, Mathematics for Computer Science 2018.
4. Huth, Michael and Ryan, Mark. Logic in Computer Science: Modelling and Reasoning about Systems. 2nd Edition, Cambridge University Press, 2004.
5. MIT OpenCourseWare. Mathematics for Computer Science (Course 6.1200J). 2024.
6. Mileti, Joseph. Modern Mathematical Logic. Cambridge University Press, 2023.
7. Schöning, Uwe. Logic for Computer Scientists. Birkhauser Classics.
8. Yu. I. Manin, A Course in Mathematical Logic for Mathematicians, Second Edition, 2010.
9. Elliott Mendelson, Introduction to Mathematical Logic, 4th edition, 1996.
10. Yagoub Mohamed Amine, Introduction à la logique mathématique, Université d'ELOUED, 2024.

Tutorials (TDs)

- TD 1: Introduction to Logic and Reasoning
- TD 2: Syntax and Semantics of Propositional Logic
- TD 3: Atomic vs. Compound Propositions, Translation, and Trees
- TD 4: WFFs and Substitution Mechanics
- TD 5: Syntax Trees and Structural Properties
- TD 6: Semantic Valuations and Truth Tables
- TD 7: Classification and Semantic Equivalence
- TD 8: Normal Forms and Logical Models
- TD 9: Refutation and Clausal Logic
- TD 10: Applying the Resolution Method
- TD 11: From Propositional Resolution to First-Order Logic
- TD 12: Predicate Logic - Syntax and Semantic Structures

TD 1: Introduction to Logic and Reasoning

Exercises

Exercise 1: Defining Propositions

Classify the following as a Proposition (and give its truth value) or a Non-Proposition (explaining if it is a question, command, open sentence, or paradox):

1. The number 24 is even.
2. Stop talking!
3. $x + 5 = 10$.
4. The sun rises in the West.
5. Is it cold outside?
6. This statement is false.
7. 7 is a prime number.
8. Close the window.

Exercise 2: Compound Truth Values

Determine the truth value of the following complex propositions. Show your reasoning:

1. $2 + 2 = 4$ and $5 < 1$.
2. 10 is divisible by 2 or 9 is a prime number.
3. $0 < -1$.

Exercise 3: Intuitive vs. Formal Validity; Generalization Fallacies

Analyze if the conclusion logically follows from the premises. Explain why:
Argument: Premise 1: If I am in Algiers, I am in Algeria. Premise 2: I am in Algeria. Conclusion: Therefore, I am in Algiers.

Analyze the following argument:

Premise 1: Some students are hard-working.
Premise 2: Mourad is a student.
Conclusion: Mourad is hard-working.

Does the conclusion necessarily follow? Why or why not?

Exercise 4: Deductive Syllogisms

Consider the following classic argument:

Premise 1: All humans are mortal.
Premise 2: Socrates is a human.
Conclusion: Socrates is mortal.

Is this argument valid? Contrast this with the argument in Exercise 4.

Exercise 5: Structure of an Argument; Validity vs. Factual Truth

Using the argument: *"All even numbers are divisible by 2. 8 is an even number. Therefore, 8 is divisible by 2."*

1. Formally identify the Premises and the Conclusion.
2. Explain why this argument is considered "Valid."

Determine the validity of this argument:

remise 1: All mammals can fly.
Premise 2: Whales are mammals.
Conclusion: Whales can fly.

Explain how an argument can be valid even if its conclusion is false.

Exercise 6: Formalizing Language

Let $P =$ "The user enters the password" and $Q =$ "The system grants access." Translate the following into symbolic logic $(\neg, \wedge, \vee, \Rightarrow)$:

- The system grants access although the user did not enter the password.
- If the user does not enter the password, the system does not grant access.
- The user enters the password or the system does not grant access.

Exercise 7: Logical Puzzles and Definitions

The Paradox: Ali says "I am a Knave." If Knights always tell the truth and Knaves always lie, what is Ali?
Definitions: In your own words, define the difference between *Truth* and *Validity*.

TD 2: Syntax and Semantics of Propositional Logic

Exercises

Exercise 1: Conceptual Foundations

For each of the following statements, indicate whether it concerns Syntax or Semantics:

"This expression is missing a closing parenthesis."
"This formula is false under the given valuation v."
"The symbol $\wedge$ must be placed between two formulas."
"The formula evaluates to True (1)."
"A syntactically correct formula follows the formation rules of the language."

Exercise 2: Identifying Well-Formed Formulas (WFFs)

Determine which of the following are Well-Formed Formulas (WFFs). If a formula is incorrect, specify the rule violated:

1. $P \vee \wedge Q$
2. $(\neg(P \Rightarrow Q))$
3. $((P \wedge Q) \vee R) \Rightarrow S$
4. $\neg\neg\neg P$
5. $P(Q \vee R)$
6. $\wedge PQ$
7. $\neg(P\vee)$

Exercise 3: The Importance of Parentheses (Structure)

In propositional logic, parentheses determine the Main Connective. Consider the sequence of symbols: $P \wedge Q \vee R$.

1. Draw/represent the syntax tree for $(P \wedge Q) \vee R$ and $P \wedge (Q \vee R)$.
2. Explain how the truth conditions (meaning) change between the two versions.

Exercise 4: Substitution

Given the formula $\phi = (P \Rightarrow Q)$, perform the following substitution:

1. Substitute the atom Q with the formula $(R \wedge S)$.
2. Is the resulting formula still a WFF?

Exercise 5: Semantic Interpretations (Valuations)

Let the interpretation (valuation) v be: $v(P) = 1$ (True), $v(Q) = 0$ (False), and $v(R) = 1$ (True). Calculate the truth value of the following formulas:

- $(P \wedge Q) \Rightarrow R$
- $\neg P \vee (Q \Leftrightarrow R)$
- $(P \Rightarrow Q) \wedge (Q \Rightarrow R)$

Exercise 6: Formalization of Natural Language

Translate these sentences into Propositional Logic formulas. Define your atoms (e.g., $P = \dots$) clearly.

- "If it is not a holiday and it is a weekday, then I must go to work."
- "Access is granted if and only if the user has a valid ID or is an administrator."
- "Neither the monitor nor the keyboard is plugged in."

Exercise 7: Short Answer - Theory

Explain the technical difference between a syntactically correct formula and a semantically valid formula (tautology).

TD 3: Atomic vs. Compound Propositions, Translation, and Trees

Exercises

Exercise 1: Atomic or Compound?

Classify each sentence as Atomic or Compound. If it is compound, identify the atomic parts and the main connective:

1. "The processor is overheating."
2. "If the battery is low, then the screen brightness decreases."
3. "The user is authenticated and has admin privileges."
4. "x is not a prime number."
5. "A triangle is equilateral if and only if its angles are all 60 degrees."
6. "The system is online."

Exercise 2: Subformulas and Structure

For the complex formula $\phi = ((P \Rightarrow Q) \wedge (\neg R \vee S))$:

1. List all unique subformulas of ϕ.
2. Draw a syntax tree for the formula $\psi = ((P \wedge \neg Q) \Rightarrow R)$.

Exercise 3: Advanced Formalization

Define your atoms (e.g., P: "I study", Q: "I pass") and translate the following natural language sentences into propositional formulas:

1. "I will pass the exam only if I study."
2. "I will not pass the exam unless I study."
3. "Studying is a necessary condition for passing the exam."
4. "It is not the case that I study but do not pass."
5. "Either the power is out, or the monitor is broken and the cable is unplugged."
6. "The server is down or the network is unavailable."

Exercise 4: Truth Tables

Construct the truth table for the formula $\phi = (P \wedge \neg Q)$.

Exercise 5: Semantic Logic Gates

In Computer Science, logical connectives are visualized as physical gates.

1. Describe the components of a circuit that represents the formula $(P \wedge Q) \vee \neg R$.
2. Explain the difference between the "Inclusive OR" ($\vee$) used in logic and the "Exclusive OR" (XOR) often used in common English.

Exercise 6: Case Study - The Buggy Code

A programmer writes the following conditional statement:

```
if (!(age < 18 || has_permission)) {
    allow_entry();
}
```

1. Identify the atomic propositions.
2. Translate this code condition into a logical formula.
3. Using De Morgan's Laws, find a simpler way to write this condition in both English and code.

TD 4: WFFs and Substitution Mechanics

Exercise 1: Validating WFFs

Identify which of the following strings are Well-Formed Formulas (WFFs). If they are not, specify the rule violated (e.g., incomplete, missing parentheses, or incorrect operator placement):

1. $P \Rightarrow (Q \Rightarrow R)$
2. $P\neg Q$
3. $(\wedge PQ)$
4. $((P \vee Q) \wedge \neg R) \Leftrightarrow S$
5. $P \Rightarrow\Rightarrow Q$
6. $P\wedge$
7. $(P \vee Q$
8. $P \vee \neg \wedge Q$

Exercise 2: Working with Priority

Rewrite the following formulas by adding all "invisible" parentheses based on the standard priority hierarchy ($\neg > \wedge > \vee > \Rightarrow > \Leftrightarrow$):

1. $\neg P \wedge Q \Rightarrow R$
2. $P \vee Q \wedge R$
3. $P \Rightarrow Q \Leftrightarrow \neg R$

Exercise 3: Variables and Interpretations

Given the formula $\phi = (P \wedge Q) \Rightarrow (\neg R \vee S)$:

- List all propositional variables in ϕ.
- Calculate the total number of possible interpretations (truth table rows) for this formula. (Recall: 2^n where n is the number of variables).

Exercise 4: Substitution Mechanics

Perform the following substitutions on the given formulas and write the resulting string:

Let $G = (A \wedge B) \Rightarrow \neg C$. Find $G[A \leftarrow (P \Leftrightarrow Q)]$.
Let $G = (A \wedge B) \Rightarrow \neg C$. Find $G[C \leftarrow \neg A]$.
Let $\phi = P \Rightarrow Q$. Substitute P by $(R \wedge S)$.
Let $\phi = \neg P \vee Q$. Substitute Q by $(P \wedge R)$.

Exercise 5: Syntax Trees and Sub-formulas

Draw the syntax tree for the following formulas and list their sub-formulas:

1. $\psi = (P \vee Q) \Rightarrow \neg R$
2. $\chi = (\neg P \vee Q) \vee \neg R$ (A formula containing only literals and $\vee$).

Exercise 6: Interpretation Evaluation

Calculate the truth value of the formula $\phi = (P \wedge \neg Q) \vee R$ under the following valuation I:

$$I(P) = T, \quad I(Q) = T, \quad I(R) = F$$

TD 5: Syntax Trees and Structural Properties

Exercise 1: Well-Formed Formulas and Main Connectives

For each of the following expressions, determine if it is a Well-Formed Formula (WFF). If it is, identify the Main Connective. If not, explain why.

1. $\neg(P \wedge Q)$
2. $(P \Rightarrow (Q\vee))$
3. $((P \Rightarrow \neg Q) \vee (R \wedge P))$
4. $\neg\neg P$
5. $((A \wedge B) \Rightarrow C) \Leftrightarrow \neg(A \vee B)$

Exercise 2: Decomposition and Subformulas

Given the formula $\phi = ((P \Rightarrow Q) \wedge (\neg R \vee S))$:
- List all atomic subformulas.
- List all unary subformulas.
- List all binary subformulas.
- Verify the total number of subformulas.

Exercise 3: Constructing Syntax Trees

Construct the full syntax tree for the following formulas. Clearly label the internal nodes (operators) and leaves (atoms).

- $\neg(P \vee Q)$
- $(P \wedge (Q \Rightarrow R))$
- $((P \Rightarrow Q) \Leftrightarrow \neg R)$
- $H = \neg(P \wedge Q) \Rightarrow (R \vee \neg P)$

Exercise 4: Reading Syntax Trees

A teacher provides a syntax tree with the following structure:

- Root: $\wedge$
- Left Child of Root: $\Rightarrow$ (with children P, Q)
- Right Child of Root: $\neg$ (with child R)

1. Write the linear WFF corresponding to this tree.
2. Identify the immediate subformulas of the whole formula.

Exercise 5: Structural Properties and Height

Consider the formula $\psi = ((P \wedge Q) \Rightarrow (R \vee \neg S))$.

1. Count the number of atomic propositions and the number of connectives.
2. Show that the total number of subformulas equals the total number of nodes in the syntax tree.
3. Calculate the Height of the tree (the number of levels from the root to the deepest leaf).

Exercise 6: Priority and Ambiguity

Consider the string $P \wedge Q \Rightarrow R \vee S$.

1. Draw the tree assuming standard priority ($\wedge, \vee$ before $\Rightarrow$). Identify the Main Connective.
2. Draw the tree assuming reverse priority ($\Rightarrow$ before $\wedge, \vee$).
3. Explain how the "meaning" changes: which version is an implication and which is a conjunction?

TD 6: Semantic Valuations and Truth Tables

Exercise 1: Step-by-Step Truth Evaluation

Let the interpretation I (or valuation v) be defined by: $I(P) = T, I(Q) = F, I(R) = T$. Compute the truth value of the following formulas, showing each intermediate step using semantic rules:

1. $\neg(P \wedge Q) \vee \neg R$
2. $(P \Rightarrow Q) \Leftrightarrow (Q \Rightarrow \neg R)$
3. $P \vee (Q \wedge R) \Rightarrow \perp$ (where $\perp$ is the constant False). $(P \wedge R) \Rightarrow Q$

Exercise 2: Recursive Evaluation and Intuition

Consider the formula $\phi = \neg(P \Rightarrow (Q \wedge R))$ under the interpretation $I(P) = T, I(Q) = T, I(R) = F$.

1. Identify the main connective.
2. Evaluate $I(\phi)$ step-by-step.
3. Intuitive Reasoning: Explain why the final truth value makes sense in natural language (e.g., "It is not the case that if...").

Exercise 3: Constructing Basic Truth Tables

Construct the full truth table for the following formulas and classify them as a Tautology, Contradiction, or Contingent:

$$F_1 = (P \Rightarrow Q) \wedge (Q \Rightarrow P)$$
$$F_2 = P \vee \neg Q$$
$$F_3 = (P \wedge Q) \Rightarrow P$$

Exercise 4: The 3-Variable Truth Table

Consider the formula $\phi = (P \wedge Q) \Rightarrow (R \vee \neg P)$.

1. How many rows are required for this truth table?
2. Construct the table step-by-step using intermediate columns for $(P \wedge Q)$ and $(R \vee \neg P)$.
3. Identify all specific interpretations for which ϕ is False.
4. Is ϕ a tautology? Justify your answer.

Exercise 5: The "Broken" Implication (Case Study)

A student argues that the statement *"If $1 + 1 = 3$, then I am the King of France"* must be False because the premise is false.

1. Using the truth table for $A \Rightarrow B$, explain why this sentence is logically True (1).
2. Define the concept of "Vacuous Truth."

Exercise 6: Semantic Equivalence and Transitivity

1. Using truth tables, prove the logical equivalence: $(P \Rightarrow Q) \equiv (\neg P \vee Q)$.
2. Construct a truth table to determine if the "Transitivity of Implication" is a tautology:

$$((P \Rightarrow Q) \wedge (Q \Rightarrow R)) \Rightarrow (P \Rightarrow R)$$

TD 7: Classification and Semantic Equivalence

Exercise 1: Classification via Truth Tables

Construct truth tables for each formula and classify them as a Tautology ($\top$), Antilogy (Contradiction $\bot$), or Contingency:

1. $(P \wedge (P \Rightarrow Q)) \Rightarrow Q$ (This is the rule of *Modus Ponens*).
2. $(P \vee Q) \wedge (\neg P \wedge \neg Q)$
3. $P \Rightarrow (Q \Rightarrow P)$
4. $(P \Rightarrow Q) \vee (Q \Rightarrow P)$
5. $(P \wedge Q) \wedge (\neg P \vee \neg Q)$

Exercise 2: Proving Semantic Equivalence

Two formulas are equivalent ($\equiv$) if their final truth table columns are identical. Prove the following standard logical laws:

1. Contraposition: $(P \Rightarrow Q) \equiv (\neg Q \Rightarrow \neg P)$
2. Absorption: $P \wedge (P \vee Q) \equiv P$
3. Exportation: $(P \wedge Q) \Rightarrow R \equiv P \Rightarrow (Q \Rightarrow R)$
4. De Morgan's Law: $\neg(P \vee Q) \equiv (\neg P \wedge \neg Q)$

Exercise 3: Simplification and Transformation

Using the laws of logic (Double Negation, De Morgan, etc.) instead of truth tables:

1. Simplify the expression: $\neg(\neg P \vee Q) \vee \neg P$.
2. Rewrite $(P \wedge Q) \Rightarrow R$ using only the symbols $\{\neg, \vee\}$.
3. Find a simplified formula equivalent to $\neg(P \Rightarrow Q)$.

Exercise 4: Logical Reasoning in Coding (XOR)

A developer wrote the following check for a security gate:

```
if ( (UserHasKey || UserHasPassword) && !(UserHasKey && UserHasPassword) )
```

1. Translate this into a formal logic formula using K for Key and P for Password.
2. Construct a truth table for this formula. Is it a tautology, antilogy, or contingency?
3. In plain English, what condition is the developer checking for?

Exercise 5: Argument Validity

In logic, an argument "Premise 1, Premise 2, Therefore Conclusion" is valid if the formula ((P1 $\wedge$ P2) $\Rightarrow$ Conclusion) is a Tautology.
Test the validity of: "If P implies Q, and P is true, then Q must be true."

TD 8: Normal Forms and Logical Models

Exercise 1: Identifying Normal Forms

Identify if the following formulas are in CNF (Conjunctive Normal Form), DNF (Disjunctive Normal Form), Both, or Neither:

1. $(P \wedge Q) \vee (\neg P \wedge R)$
2. $(P \vee Q) \wedge (\neg P \vee R) \wedge S$
3. $P \vee Q \vee \neg R$
4. $(P \Rightarrow Q) \wedge (Q \vee R)$
5. $(P \wedge Q) \vee R$
6. $(P \vee Q) \wedge R$

Exercise 2: Transformation via Truth Tables

Given the formula $F = (P \Rightarrow Q) \Leftrightarrow R$:

1. Construct the full truth table ($2^3 = 8$ rows).
2. Extract the Full DNF by identifying the rows where $F = 1$ (Sum of Products).
3. Extract the Full CNF by identifying the rows where $F = 0$ (Product of Sums).

Exercise 3: Satisfaction and Models

Consider the formula $G = (P \vee Q) \Rightarrow (P \wedge \neg R)$.

1. Definition: What is a "Model" in propositional logic?
2. Find three different interpretations that are Models of G.
3. Find one interpretation that is not a model of G (a counter-model).
4. Is G satisfiable? Is it valid? Explain based on your findings.

Exercise 4: Boolean Algebra Transformations

Convert the following formulas into the specified normal form using logical laws (De Morgan, Distributivity, Double Negation) *without* using a truth table:

1. $H_1 = \neg((P \wedge Q) \vee \neg R) \quad \rightarrow$ Convert to CNF.
2. $H_2 = (P \Rightarrow Q) \vee R \quad \rightarrow$ Convert to CNF.
3. $H_3 = P \wedge (Q \vee R) \quad \rightarrow$ Convert to DNF.

Exercise 5: Reasoning and Satisfiability

Determine if the following formula is Satisfiable by attempting to find at least one model:

$$\phi = (P \vee Q) \wedge (\neg P \vee Q) \wedge (P \vee \neg Q)$$

TD 9: Refutation and Clausal Logic

Exercise 1: Clause Identification and Syntax

1. Identification: Which of the following are valid *clauses*? Explain why:
   - $P \lor \neg Q$
   - $P \land Q$
   - $\neg R$
   - $(P \lor Q) \land R$
2. Notation: Convert these formulas into Clause Set notation (e.g., $\{\{L_1, L_2\}, \{L_3\}\}$):
   - $(P \lor Q) \land (\neg R \lor S) \land P$
   - $(\neg A \lor B \lor C) \land (A \lor \neg C)$
   - $P \land \neg P$

Exercise 2: Satisfaction and Consistency

1. Evaluation: Does the interpretation $I(P) = T, I(Q) = F$ satisfy the formula $\phi = (P \lor Q) \land \neg Q$?
2. Satisfiability: Determine whether the set $\Gamma = \{P \lor Q, \neg P, \neg Q\}$ is satisfiable.
3. Detection: Is the clause set $\{P, \neg P \lor Q, \neg Q\}$ consistent? Justify your answer.

Exercise 3: Step-by-Step Clausal Transformation

Transform the following formulas into Clause Sets. Show every intermediate step (Implication removal, De Morgan, Double Negation, and Distributivity):

1. $F_1 = \neg((P \land Q) \Rightarrow R)$
2. $F_2 = P \Leftrightarrow Q$
3. $F_3 = \neg(\neg P \land (Q \lor R))$
4. $F_4 = (P \Rightarrow Q) \land \neg R$

Exercise 4: The Logic of Refutation

Suppose you want to prove that "If it rains (R), then the ground is wet (W)" is valid using the Refutation Method.

1. Identify the Premise and the Conclusion.
2. What is the negated conclusion?
3. Write the set of formulas you would need to prove are unsatisfiable.
4. Theory: What does the "empty clause" ($\Box$) represent in clausal logic?

Exercise 5: Advanced Argument Formalization

Formalize the following English argument and convert it into a single Clause Set for a refutation proof: *"If the file is open (O), the lock is active (L). The file is open. Prove: The lock is active."*

TD 10: Applying the Resolution Method

Exercise 1: Basic Resolution Steps

A resolvent is a new clause derived from two '“parent” clauses that contain a complementary pair of literals (e.g., P and $\neg P$). Find all possible resolvents for the following pairs:

1. $C_1 = \{P, Q, R\}, \quad C_2 = \{\neg Q, S\}$
2. $C_1 = \{P, \neg Q\}, \quad C_2 = \{Q\}$
3. $C_1 = \{P, Q\}, \quad C_2 = \{\neg P, R\}$
4. $C_1 = \{\neg P, \neg R\}, \quad C_2 = \{P, R\}$ *(Note: Can you resolve both at once?)*

Exercise 2: Resolution Chains and the Empty Clause

Using the resolution rule, derive the final conclusion for the following sets. If you reach the empty clause ($\square$), state what this implies about the set.

1. $\Gamma = \{P, \quad \neg P \vee Q, \quad \neg Q \vee R\}$ (Derive the new clause)
2. $\Sigma = \{P \vee Q, \quad \neg P, \quad \neg Q\}$ (Resolve until the end)

Exercise 3: Proving Validity by Resolution

To prove an argument is valid ($Premises \models Conclusion$), we must negate the conclusion and show the resulting Clause Set leads to $\square$. Prove:

1. Syllogism: $\{P \Rightarrow Q, \quad Q \Rightarrow R\} \models P \Rightarrow R$
2. Case Analysis: $\{(P \vee Q), \quad (P \Rightarrow R), \quad (Q \Rightarrow R)\} \models R$
3. Simple Tautology: Show $(P \wedge Q) \Rightarrow Q$ is valid using resolution.

Exercise 4: Identifying Logical Fallacies

Try to use resolution to prove: $\{P \Rightarrow Q, \quad Q\} \models P$.

- Observe what happens during the resolution process. Can you derive $\square$?
- What does this result tell you about the “Fallacy of Affirming the Consequent”?

Exercise 5: Visualization and Strategy

1. Resolution Tree: Draw a tree for $S = \{\{P, Q\}, \{\neg P, Q\}, \{P, \neg Q\}, \{\neg P, \neg Q\}\}$ showing how it reduces to $\square$.
2. Conceptual: Why can different resolution “orders” (which clauses you pick first) lead to different proof speeds?
3. Incompleteness: What does it mean for the original formula if you have exhausted all possible resolutions but never derived $\square$?

TD 11: From Propositional Resolution to First-Order Logic

Part A: Advanced Resolution Refutation

Exercise 1: Complete Resolution Proofs

Use the Resolution Method to determine the validity or satisfiability of the following:

1. Validity: Prove that $(P \wedge (P \Rightarrow Q)) \Rightarrow Q$ is valid. (Step: Negate, convert to clauses, and resolve).
2. Satisfiability: Is the set $\Gamma = \{(P \vee Q), (\neg P \vee Q), (\neg Q \vee R)\}$ satisfiable? Attempt to find a model or derive $\square$.
3. Inconsistency: Show that the set $\{P \vee Q, \neg P, \neg Q\}$ is inconsistent using a resolution tree.

Exercise 2: Resolution Theory

1. Why must we negate a formula before applying the resolution method to prove its validity?
2. Can resolution prove satisfiability *directly*? If the resolution process ends without finding $\square$, what does that imply?

—

Part B: Introduction to First-Order Logic (FOL)

Exercise 3: Identifying FOL Syntax

Label each of the following components as a Constant, Variable, Function, or Predicate:

1. x
2. $Teacher(x)$
3. $john$
4. $Successor(n)$
5. $GreaterThan(x, 5)$

Exercise 4: Translating to Predicate Logic

Let $S(x)$ be "x is a student," $C(x)$ be "x is clever," and $L(x, y)$ be "x likes y." Translate the following into FOL formulas:

1. All students are clever.
2. Some students are clever.
3. No student is clever.
4. Every student likes some clever person.
5. There is a student who likes everyone.

Exercise 5: Nested Quantifiers and Domains

1. Quantifier Order: Explain the difference in meaning between $\forall x \exists y L(x, y)$ and $\exists y \forall x L(x, y)$.
2. Domain of Discourse: Consider the formula $\forall x(x^2 \geq 0)$. Is this True or False if the domain is:
   - The set of Real Numbers ($\mathbb{R}$)?
   - The set of Complex Numbers ($\mathbb{C}$)?
3. Why must we define the "Domain" before evaluating a quantified formula?

TD 12: Predicate Logic - Syntax and Semantic Structures

Exercise 1: Syntax Analysis (Terms, Formulas, and Scope)

In First-Order Logic, we distinguish between *objects* (terms) and *statements* (formulas).

1. Identify Terms: Which of the following are terms?
   - x
   - $f(x)$
   - $P(x)$
   - $add(x, y)$
2. Identify Atomic Formulas: Which of the following are atomic formulas?
   - $Student(x)$
   - $x > y$
   - $\forall x P(x)$
3. Scope and Variables: For each formula, identify the Scope of the quantifier and state whether each occurrence of a variable is Free or Bound:
   - $\phi_1 = \forall x(P(x, y) \wedge Q(x))$
   - $\phi_2 = (\exists x P(x)) \Rightarrow R(x)$
   - $\phi_3 = \exists x(P(x) \wedge Q(x)) \vee R(x)$
   - $\phi_4 = \forall x \exists y(L(x, y) \Rightarrow \forall z M(z))$

Exercise 2: Translation into Logic

Translate the following English sentence into a formal FOL formula using appropriate predicates:

> "There exists a student who has passed all exams."

Exercise 3: Working with Structures

Consider a structure $\mathcal{M}$ where the Domain $D = \{1, 2, 3\}$. Let $P(x)$ mean "x is even" and $Q(x, y)$ mean "$x < y$." Determine the truth value (True/False) of the following:

1. $\exists x P(x)$
2. $\forall x(P(x) \vee \neg P(x))$
3. $\forall x \exists y Q(x, y)$
4. $\exists y \forall x(x \neq y \Rightarrow Q(x, y))$
5. $\exists x(x > 2)$

Exercise 4: Counter-Models and Validity

To prove a formula is not valid, you must find a structure (counter-model) where the premise is true but the conclusion is false. Find a counter-model for:

$$\forall x(P(x) \vee Q(x)) \Rightarrow (\forall x P(x) \vee \forall x Q(x))$$

(Hint: Consider a domain of integers where P is "even" and Q is "odd").

Exercise 5: The Role of the Domain

Evaluate the formula $\forall x \exists y(x + y = 0)$ in the following contexts:

1. Is it True in the domain of Integers ($\mathbb{Z}$)?
2. Is it True in the domain of Natural Numbers ($\mathbb{N} = \{0, 1, 2, \dots\}$)?
3. Discussion: What does this tell us about the relationship between Logic and the "World" (Domain) we choose to describe?