Discrete Structure/Mathematics

Dr. Qaiser Abbas
Department of Computer Science & IT,
University of Sargodha, Sargodha, 40100, Pakistan
qaiser.abbas@uos.edu.pk

1.2 Applications of Propositional Logic

- **Introduction**
  - Logic has many important applications to mathematics, computer science, and numerous other disciplines.
  - Statements in mathematics and the sciences and in natural language often are imprecise or ambiguous. To make such statements precise, they can be translated into the language of logic.
  - Some of applications of propositional logic are discussed next.

- **Translating English Sentences**
  - Translating sentences into compound statements removes the ambiguity in natural language sentences.
  - Once we have translated sentences from English into logical expressions, we can analyze these logical expressions to determine their truth values, we can manipulate them, and we can use rules of inference (Section 1.6) to reason about them.
  - To illustrate the process of translating an English sentence into a logical expression, consider the Example.

– EXAMPLE

- How can this English sentence be translated into a logical expression?

  "You can access the Internet from campus only if you are a computer science major or you are not a freshman."

- *Solution:*
  - There are many ways to translate this sentence into a logical expression. we will use propositional variables to represent each sentence part and determine the appropriate logical connectives between them. We let a, c, and f represent "You can access the Internet from campus," "You are a computer science major," and "You are a freshman," respectively. Noting that "only if" is one way a conditional statement can be expressed, this sentence can be represented as
  - $a \rightarrow (c \vee \neg f)$.
- Of course, there are other ways to represent the original sentence as a logical expression.

- **System Specifications**
  - System and software engineers take requirements in natural language and produce precise and unambiguous specifications that can be used as the basis for system development. The following example shows how compound propositions can be used in this process.
  - System specifications should be **consistent**, that is, they should not contain conflicting requirements that could be used to derive a contradiction. When specifications are not consistent, there would be no way to develop a system that satisfies all specifications.

- **EXAMPLE**
- Determine whether these system specifications are consistent:
  - "The diagnostic message is stored in the buffer or it is retransmitted."
  - "The diagnostic message is not stored in the buffer."
  - "If the diagnostic message is stored in the buffer, then it is retransmitted."
- *Solution:*
  - We first express them using logical expressions. Let p denote "The diagnostic message is stored in the buffer" and let q denote "The diagnostic message is retransmitted." The specifications can then be written as $p \vee q$, $\neg p$, and $p \rightarrow q$.
  - An assignment of truth values that makes all three specifications true must have p false to make $\neg p$ true.
  - Because we want $p \vee q$ to be true but p must be false, q must be true.
  - Because $p \rightarrow q$ is true when p is false and q is true
  - We conclude that these specifications are consistent, because they are all true when p is false and q is true.

- **Boolean Searches**
  - Logical connectives are used extensively in searches of large collections of information, such as indexes of Web pages.
  - In Boolean searches, the connective *AND* is used to match records that contain both of two search terms, the connective *OR* is used to match one or both of two search terms, and the connective *NOT* (sometimes written as *AND NOT*) is used to exclude a particular search term.
  - Note: Read it yourself

- **Logic Puzzles**
  - Puzzles that can be solved using logical reasoning are known as **logic puzzles**.
  - Solving logic puzzles is an excellent way to practice working with the rules of logic.
  - **EXAMPLE**
    - In [Sm78] Smullyan posed many puzzles about an island that has two kinds of inhabitants, *knights*, who always tell the truth, and their opposites, *knaves*, who always lie. You encounter two people A and B. What are A and B if A says "B is a knight" and B says "The two of us are opposite types?"

– *Solution:*

- Let p and q be the statements that A is a knight and B is a knight, respectively, so that ¬p and ¬q are the statements that A is a knave and B is a knave, respectively.
- We first consider the possibility that A is a knight; this is the statement that p is true.
  - If A is a knight, then he is telling the truth when he says that B is a knight, so that q is true, and A and B are the same type.
  - However, if B is a knight, then B's statement that A and B are of opposite types, the statement $(p \wedge \neg q) \vee (\neg p \wedge q)$, would have to be true, which it is not, because A and B are both knights.
  - Consequently, we can conclude that A is not a knight, that is, that p is false.

- Now, we consider the other possibility that A is knave;
  - If A is a knave, then because everything a knave says is false, A's statement that B is a knight, that is, that q is true, is a lie. This means that q is false and B is also a knave.
  - Furthermore, if B is a knave, then B's statement that A and B are opposite types is a lie, which is consistent with both A and B being knaves. We can conclude that both A and B are knaves.

– **Muddy children puzzle**

- **Read it yourself in Example 8**

- **Logic Circuits**
  - Propositional logic can be applied to the design of computer hardware. This was first observed in 1938 by Claude Shannon in his MIT master's thesis.
  - A **logic circuit** (or **digital circuit**) receives input signals p1, p2, . . . , pn, each a bit [either 0 (off) or 1 (on)], and produces output signals s1, s2, . . . , sn, each a bit.
  - Complicated digital circuits can be constructed from three basic circuits, called **gates**, shown in Figure 1 next.

- The **inverter**, or **NOT gate**, takes an input bit p, and produces as output ¬p.
- The **OR gate** takes two input signals p and q, each a bit, and produces as output the signal p ∨ q.
- Finally, the **AND gate** takes two input signals p and q, each a bit, and produces as output the signal p ∧ q.
- We use combinations of these three basic gates to build more complicated circuits, such as that shown in Figure 2.

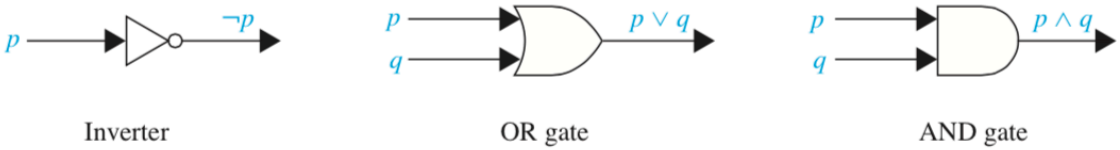

FIGURE 1 **Basic logic gates.**

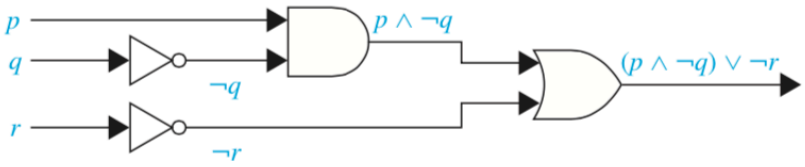

FIGURE 2 **A combinatorial circuit.**

- **EXAMPLE**
  - Build a digital circuit that produces the output (p ∨ ¬r) ∧ (¬p ∨ (q ∨ ¬r)) when given input bits p, q, and r.
  - Solution:

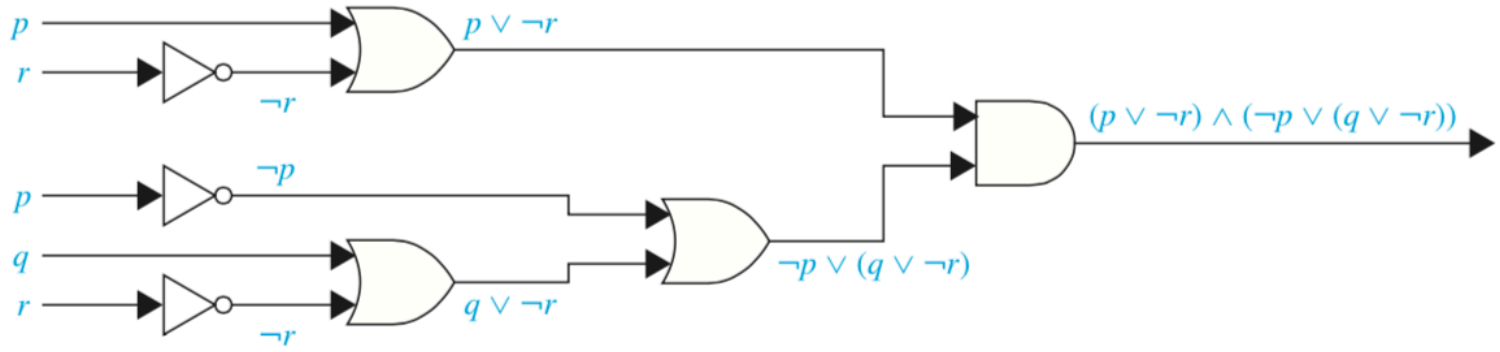

FIGURE 3 The circuit for $(p \vee \neg r) \wedge (\neg p \vee (q \vee \neg r))$.

Assignment 1.2

- Exercise No's, 6, 10, 16, 24, 32, 38, 43
- Deadline is the next lecture

1.3 Propositional Equivalences

- **Introduction**
  - An important type of step used in a mathematical argument is the replacement of a statement with another statement with the same truth value.
  - **DEFINITIONS**
    - A compound proposition that is always true, no matter what the truth values of the propositional variables that occur in it, is called a *tautology*.
    - A compound proposition that is always false is called a *contradiction*.
    - A compound proposition that is neither a tautology nor a contradiction is called a *contingency*.

– **EXAMPLE**

– We can construct examples of tautologies and contradictions using just one propositional variable.

– Consider the truth tables of p ∨ ¬p and p ∧ ¬p, shown in Table 1. Because p ∨ ¬p is always true, it is a tautology. Because p ∧ ¬p is always false, it is a contradiction.

TABLE 1 Examples of a Tautology and a Contradiction.

p	$\neg p$	$p \vee \neg p$	$p \wedge \neg p$
T	F	T	F
F	T	T	F

- **Logical Equivalences**
  - Compound propositions that have the same truth values in all possible cases are called **logically equivalent**.
  - **DEFINITION**
    - The compound propositions p and q are called *logically equivalent* if $p \leftrightarrow q$ is a tautology. The notation $p \equiv q$ denotes that p and q are logically equivalent.
    - ***Remark:*** The symbol ≡ is not a logical connective, and $p \equiv q$ is not a compound proposition but rather is the statement that $p \leftrightarrow q$ is a tautology. The symbol ⇔ is sometimes used instead of ≡ to denote logical equivalence.

– EXAMPLE

- Show that one of the De Morgan's Laws in Table 2 are equivalent.
- ***Solution:*** The truth tables for these compound propositions are displayed in Table 3. Because the truth values of the compound propositions ¬(p ∨ q) and ¬p ∧ ¬q agree for all possible combinations of the truth values of p and q , it follows that ¬(p ∨ q) ↔ (¬p ∧ ¬q) is a tautology and that these compound propositions are logically equivalent.

TABLE 2 De Morgan's Laws.

$$\neg(p \wedge q) \equiv \neg p \vee \neg q$$

$$\neg(p \vee q) \equiv \neg p \wedge \neg q$$

TABLE 3 Truth Tables for $\neg(p \vee q)$ and $\neg p \wedge \neg q$.

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

- Table 6 contains some important equivalences. In these equivalences, **T** denotes the compound proposition that is always true and **F** denotes the compound proposition that is always False
- We also display some useful equivalences for compound propositions involving conditional statements and bi-conditional statements in Tables 7 and 8, respectively.

TABLE 6 Logical Equivalences.

Equivalence	*Name*
$p \wedge \mathbf{T} \equiv p$ $p \vee \mathbf{F} \equiv p$	Identity laws
$p \vee \mathbf{T} \equiv \mathbf{T}$ $p \wedge \mathbf{F} \equiv \mathbf{F}$	Domination laws
$p \vee p \equiv p$ $p \wedge p \equiv p$	Idempotent laws
$\neg(\neg p) \equiv p$	Double negation law
$p \vee q \equiv q \vee p$ $p \wedge q \equiv q \wedge p$	Commutative laws
$(p \vee q) \vee r \equiv p \vee (q \vee r)$ $(p \wedge q) \wedge r \equiv p \wedge (q \wedge r)$	Associative laws
$p \vee (q \wedge r) \equiv (p \vee q) \wedge (p \vee r)$ $p \wedge (q \vee r) \equiv (p \wedge q) \vee (p \wedge r)$	Distributive laws
$\neg(p \wedge q) \equiv \neg p \vee \neg q$ $\neg(p \vee q) \equiv \neg p \wedge \neg q$	De Morgan's laws
$p \vee (p \wedge q) \equiv p$ $p \wedge (p \vee q) \equiv p$	Absorption laws
$p \vee \neg p \equiv \mathbf{T}$ $p \wedge \neg p \equiv \mathbf{F}$	Negation laws

TABLE 7 **Logical Equivalences Involving Conditional Statements.**

$$p \to q \equiv \neg p \vee q$$

$$p \to q \equiv \neg q \to \neg p$$

$$p \vee q \equiv \neg p \to q$$

$$p \wedge q \equiv \neg(p \to \neg q)$$

$$\neg(p \to q) \equiv p \wedge \neg q$$

$$(p \to q) \wedge (p \to r) \equiv p \to (q \wedge r)$$

$$(p \to r) \wedge (q \to r) \equiv (p \vee q) \to r$$

$$(p \to q) \vee (p \to r) \equiv p \to (q \vee r)$$

$$(p \to r) \vee (q \to r) \equiv (p \wedge q) \to r$$

TABLE 8 **Logical Equivalences Involving Biconditional Statements.**

$$p \leftrightarrow q \equiv (p \to q) \wedge (q \to p)$$

$$p \leftrightarrow q \equiv \neg p \leftrightarrow \neg q$$

$$p \leftrightarrow q \equiv (p \wedge q) \vee (\neg p \wedge \neg q)$$

$$\neg(p \leftrightarrow q) \equiv p \leftrightarrow \neg q$$

- **Using De Morgan's Laws**
  - The two logical equivalences known as De Morgan's laws are particularly important. They tell us how to negate conjunctions and how to negate disjunctions.

TABLE 2 De Morgan's Laws.
$\neg(p \wedge q) \equiv \neg p \vee \neg q$
$\neg(p \vee q) \equiv \neg p \wedge \neg q$

  - **EXAMPLE**
    - Use De Morgan's laws to express the negations of "Miguel has a cellphone and he has a laptop computer" and "Heather will go to the concert or Steve will go to the concert."
    - Solution: Read it yourself on page 29 of Textbook

- **Constructing New Logical Equivalences**
  - The logical equivalences in Table 6, as well as any others that have been established (such as those shown in Tables 7 and 8), can be used to construct additional logical equivalences.
  - We will use the fact that if p and q are logically equivalent and q and r are logically equivalent, then p and r are logically equivalent.
  - **EXAMPLE**
    - Show that $\neg(p \vee (\neg p \wedge q))$ and $\neg p \wedge \neg q$ are logically equivalent by developing a series of logical equivalences.

– *Solution:* We will use one of the equivalences in Table 6 at a time, starting with ¬(p ∨ (¬p ∧ q)) and ending with ¬p ∧ ¬q.

$$
\begin{aligned}
\neg(p \vee (\neg p \wedge q)) &\equiv \neg p \wedge \neg(\neg p \wedge q) && \text{by the second De Morgan law}\\
&\equiv \neg p \wedge [\neg(\neg p) \vee \neg q] && \text{by the first De Morgan law}\\
&\equiv \neg p \wedge (p \vee \neg q) && \text{by the double negation law}\\
&\equiv (\neg p \wedge p) \vee (\neg p \wedge \neg q) && \text{by the second distributive law}\\
&\equiv \mathbf{F} \vee (\neg p \wedge \neg q) && \text{because } \neg p \wedge p \equiv \mathbf{F}\\
&\equiv (\neg p \wedge \neg q) \vee \mathbf{F} && \text{by the commutative law for disjunction}\\
&\equiv \neg p \wedge \neg q && \text{by the identity law for } \mathbf{F}
\end{aligned}
$$

– Consequently ¬(p ∨ (¬p ∧ q)) and ¬p ∧ ¬q are logically equivalent.

- **Propositional Satisfiability**
  - A compound proposition is **satisfiable** if there is an assignment of truth values to its variables that makes it true.
  - When no such assignments exists, that is, when the compound proposition is false for all assignments of truth values to its variables, the compound proposition is **unsatisfiable**.
  - Note that a compound proposition is unsatisfiable if and only if its negation is true for all assignments of truth values to the variables, that is, if and only if its negation is a tautology.

- When we find a particular assignment of truth values that makes a compound proposition true, we have shown that it is satisfiable; such an assignment is called a **solution** of this particular satisfiability problem.
- However, to show that a compound proposition is unsatisfiable, we need to show that *every* assignment of truth values to its variables makes it false. Although we can always use a truth table to determine whether a compound proposition is satisfiable, it is often more efficient not to, as the following Example demonstrates.
- **EXAMPLE**
  - Determine whether each of the compound propositions $(p \vee \neg q) \wedge (q \vee \neg r) \wedge (r \vee \neg p)$, $(p \vee q \vee r) \wedge (\neg p \vee \neg q \vee \neg r)$, and $(p \vee \neg q) \wedge (q \vee \neg r) \wedge (r \vee \neg p) \wedge (p \vee q \vee r) \wedge (\neg p \vee \neg q \vee \neg r)$ is satisfiable.

- *Solution:* Instead of using truth table to solve this problem, we will reason about truth values. Note that $(p \vee \neg q) \wedge (q \vee \neg r) \wedge (r \vee \neg p)$ is true when the three variable p, q, and r have the same truth value (see Exercise 40 of Section 1.1). Hence, it is satisfiable as there is at least one assignment of truth values for p, q, and r that makes it true.
- Similarly, note that $(p \vee q \vee r) \wedge (\neg p \vee \neg q \vee \neg r)$ is true when at least one of p,q, and r is true and at least one is false (see Exercise 41 of Section 1.1). Hence, $(p \vee q \vee r) \wedge (\neg p \vee \neg q \vee \neg r)$ is satisfiable, as there is at least one assignment of truth values for p, q, and r that makes it true.
- Finally, note that for $(p \vee \neg q) \wedge (q \vee \neg r) \wedge (r \vee \neg p) \wedge (p \vee q \vee r) \wedge (\neg p \vee \neg q \vee \neg r)$ to be true, $(p \vee \neg q) \wedge (q \vee \neg r) \wedge (r \vee \neg p)$ and $(p \vee q \vee r) \wedge (\neg p \vee \neg q \vee \neg r)$ must both be true. For the first to be true, the three variables must have the same truth values, and for the second to be true, at least one of three variables must be true and at least one must be false. However, these conditions are contradictory. From these observations we conclude that no assignment of truth values to p, q, and r makes $(p \vee \neg q) \wedge (q \vee \neg r) \wedge (r \vee \neg p) \wedge (p \vee q \vee r) \wedge (\neg p \vee \neg q \vee \neg r)$ true. Hence, it is unsatisfiable.

- **Applications of Satisfiability**
  - Many problems, in diverse areas such as robotics, software testing, computer-aided design, machine vision, integrated circuit design, computer networking, and genetics, can be modeled in terms of propositional satisfiability. we will show how to use propositional satisfiability to model Sudoku puzzles.
  - **SUDOKU**
    - A **Sudoku puzzle** is represented by a 9 × 9 grid made up of nine 3 × 3 subgrids, known as **blocks**, as shown in Figure 1. For each puzzle, some of the 81 cells, called **givens**, are assigned one of the numbers 1, 2, . . . , 9, and the other cells are blank.
    - The puzzle is solved by assigning a number to each blank cell so that every row, every column, and every one of the nine 3 × 3 blocks contains each of the nine possible numbers.
    - Note that instead of using a 9 × 9 grid, Sudoku puzzles can be based on $n^2 \times n^2$ grids, for any positive integer n, with the $n^2 \times n^2$ grid made up of n^2 n × n subgrids.

- Sudoku puzzles designed for entertainment have two additional important properties. First, they have exactly one solution. Second, they can be solved using reasoning alone.
- Entries in blank cells are successively determined by already known values.
- For instance, in the grid in Figure 1, the number 4 must appear in exactly one cell in the second row. How can we determine which of the seven blank cells it must appear?
- First, we observe that 4 cannot appear in one of the first three cells or in one of the last three cells of this row, because it already appears in another cell in the block each of these cells is in.
- We can also see that 4 cannot appear in the fifth cell in this row, as it already appears in the fifth column in the fourth row. This means that 4 must appear in the sixth cell of the second row.

	2	9				4		
			5			1		
	4							
				4	2			
6							7	
5								
7			3					5
	1			9				
							6	

FIGURE 1 **A 9 x 9 Sudoku puzzle.**

- Many strategies based on logic and mathematics have been devised for solving Sudoku puzzles. Here, we discuss one of the ways that have been developed for solving Sudoku puzzles with the aid of a computer, which depends on modeling the puzzle as a propositional satisfiability problem.
- To encode a Sudoku puzzle, let p(i, j, n) denote the proposition that is true when the number n is in the cell in the ith row and jth column. There are $9 \times 9 \times 9 = 729$ such propositions, as i, j, and n all range from 1 to 9.
- For example, for the puzzle in Figure 1, the number 6 is given as the value in the fifth row and first column. Hence, we see that p(5, 1, 6) is true, but p(5, j, 6) is false for j = 2,3,...,9.

– Given a particular Sudoku puzzle, we begin by encoding each of the given values. Then, we construct compound propositions that assert that every row contains every number, every column contains every number, every 3×3 block contains every number, and each cell contains no more than one number.
– It follows, as the reader should verify, that the Sudoku puzzle is solved by finding an assignment of truth values to the 729 propositions p(i,j,n) with i, j, and n each ranging from 1 to 9 that makes the conjunction of all these compound propositions true.

- For each cell with a given value, we assert $p(i, j, n)$ when the cell in row i and column j has the given value n.
- We assert that every row contains every number:

$$\bigwedge_{i=1}^{9} \bigwedge_{n=1}^{9} \bigvee_{j=1}^{9} p(i, j, n)$$

- We assert that every column contains every number:

$$\bigwedge_{j=1}^{9} \bigwedge_{n=1}^{9} \bigvee_{i=1}^{9} p(i, j, n)$$

- We assert that each of the nine 3×3 blocks contains every number:

$$\bigwedge_{r=0}^{2} \bigwedge_{s=0}^{2} \bigwedge_{n=1}^{9} \bigvee_{i=1}^{3} \bigvee_{j=1}^{3} p(3r + i, 3s + j, n)$$

- To assert that no cell contains more than one number, we take the conjunction over all values of $n, n', i,$ and j where each variable ranges from 1 to 9 and $n \neq n'$ of $p(i, j, n) \rightarrow \neg p(i, j, n')$.

We now explain how to construct the assertion that every row contains every number. First, to assert that row i contains the number n, we form $\bigvee_{j=1}^{9} p(i, j, n)$. To assert that row i contains all n numbers, we form the conjunction of these disjunctions over all nine possible values of n, giving us $\bigwedge_{n=1}^{9} \bigvee_{j=1}^{9} p(i, j, n)$. Finally, to assert that every row contains every number, we take the conjunction of $\bigwedge_{n=1}^{9} \bigvee_{j=1}^{9} p(i, j, n)$ over all nine rows. This gives us $\bigwedge_{i=1}^{9} \bigwedge_{n=1}^{9} \bigvee_{j=1}^{9} p(i, j, n)$. (Exercises 65 and 66 ask for explanations of the assertions that every column contains every number and that each of the nine 3×3 blocks contains every number.)

Given a particular Sudoku puzzle, to solve this puzzle we can find a solution to the satisfiability problems that asks for a set of truth values for the 729 variables $p(i, j, n)$ that makes the conjunction of all the listed assertions true.

3	2	9	8	1	6	4	5	7
8	6	7	5	3	4	1	9	2
1	4	5	2	7	9	6	3	8
9	3	1	7	4	2	5	8	6
6	8	4	1	5	3	2	7	9
5	7	2	9	6	8	3	1	4
7	9	6	3	2	1	8	4	5
4	1	8	6	9	5	7	2	3
2	5	3	4	8	7	9	6	1

Assignment 1.3

- Exercise No's 10, 20, 26, 30, 34, 42, 44, 58, 60, 62, 64
- Deadline is the next lecture.