Discrete Mathematics Course Notes

Felipe Balbi

January 7, 2020

Contents

1 Week 1 **7**
1.1 1.101 Introduction to discrete mathematics 7
1.2 1.104 The definition of a set 7
1.2.1 Definition of a set 7
1.2.2 Element of a set ($\in$) 8
1.2.3 Cardinality of a set (Card) 8
1.2.4 Subset of a set ($\subseteq$) 8
1.2.5 Special Sets: $\mathbb{N}$, $\mathbb{Z}$, $\mathbb{Q}$, $\mathbb{R}$ 8
1.3 1.106 The listing method and rule of inclusion 8
1.4 1.108 The powerset of a set 9
1.4.1 Powerset of a set 9
1.4.2 Cardinality of a powerset 10
1.5 1.110 Set operations 10
1.5.1 Union ($\cup$) 10
1.5.2 Intersection ($\cap$) 11
1.5.3 Difference ($-$) 12
1.5.4 Symmetric Difference ($\oplus$) 12
1.5.5 Summary 13

2 Week 2 **14**
2.1 1.201 The representation of a set using Venn diagrams 14
2.1.1 The Universal Set 14
2.1.2 Complement of a set 14
2.2 1.203 De Morgan's laws 15
2.2.1 De Morgan's First Law 15
2.2.2 De Morgan's Second Law 16
2.2.3 Proof using membership tables 16
2.3 1.205 Laws of sets: Commutative, associative and distributive . . 16
2.3.1 Commutativity 16
2.3.2 Associativity 17
2.3.3 Distributivity 17
2.3.4 Table of Set Identities 18
2.3.5 Applying set identities to simplify expressions 18
2.4 1.207 Partition 18
2.4.1 Definition of a partition of a set 19

3 Week 3 **19**
3.1 2.101 Introduction . . . 19
3.2 2.102 The Definition of A Function . . . 19
3.2.1 Definition . . . 19
3.2.2 Domain, co-domain and range of a function . . . 20
3.2.3 Image and pre-image (antecedent) of an element . . . 20
3.2.4 Example of Domain, co-domain and range . . . 20
3.3 2.104 Plotting functions . . . 21
3.3.1 Linear Functions . . . 21
3.3.2 Quadratic functions . . . 21
3.3.3 Exponential functions . . . 22
3.4 2.106 Injective and surjective functions . . . 23
3.4.1 Injective Functions . . . 23
3.4.2 Surjective Functions . . . 24

4 Week 4 **25**
4.1 2.201 Function composition . . . 25
4.2 2.203 Bijective functions . . . 26
4.2.1 Definition . . . 26
4.2.2 Exercise 1: . . . 26
4.2.3 Inverse function . . . 27
4.2.4 Exercise 2: . . . 27
4.2.5 Identity function . . . 27
4.2.6 Plotting the inverse function . . . 28
4.3 2.205 Logarithmic functions . . . 28
4.3.1 Definition . . . 28
4.3.2 Laws of logarithmic functions . . . 29
4.3.3 Exercise 1 . . . 29
4.3.4 Natural logarithm . . . 29
4.4 2.207 Floor and ceiling functions . . . 29
4.4.1 Floor function . . . 29
4.4.2 Ceiling function . . . 30
4.4.3 Exercise 1 . . . 31

5 Week 5 **31**
5.1 3.101 Introduction to propositional logic . . . 32
5.1.1 Applications of propositional logic . . . 32
5.2 3.103 Propositions . . . 32
5.2.1 Examples of propositions . . . 32
5.2.2 Propositional Variables . . . 33
5.3 3.105 Truth tables and truth sets . . . 33
5.3.1 True Tables . . . 33
5.3.2 Truth Set . . . 34
5.4 3.107 Compound propositions . . . 34
5.4.1 Negation $\neg$ . . . 35
5.4.2 Conjunction $\wedge$ . . . 35
5.4.3 Disjunction $\vee$ . . . 35
5.4.4 Exclusive-or $\oplus$ . . . 35
5.4.5 Precedence of logical operators . . . 36
5.4.6 Exercise . . . 36

6 Week 6 **37**
6.1 3.202 Logical implication ($\rightarrow$) . . . 37
6.1.1 Truth Table . . . 37
6.1.2 Different expressions for $p \rightarrow q$ . . . 37
6.1.3 Converse, inverse and contrapositive . . . 38
6.2 3.204 Logical equivalence ($\leftrightarrow$) . . . 39
6.2.1 Truth Table . . . 39
6.2.2 Equivalent propositions . . . 39
6.2.3 Proving equivalence . . . 39
6.2.4 Precedence of logical operators (Updated) . . . 40
6.3 3.206 Laws of propositional logic . . . 40

7 Week 7 **41**
7.1 4.101 Introduction to predicate logic . . . 41
7.1.1 Example 1 . . . 41
7.1.2 Example 2 . . . 41
7.2 4.103 What are predicates? . . . 41
7.2.1 Insufficiency of Propositional Logic . . . 42
7.2.2 Definition of Predicate . . . 42
7.2.3 Predicates with multiple variables . . . 42
7.2.4 Logical operations . . . 42
7.3 4.105 Quantification . . . 43
7.3.1 Universal Quantifier $\forall$ . . . 43
7.3.2 Existential Quantifier $\exists$ . . . 44
7.3.3 Uniqueness Quantifier $\exists!$ . . . 44
7.3.4 Example 1 . . . 44
7.4 4.107 Nested Quantifiers . . . 44
7.4.1 Binding Variables . . . 45
7.4.2 Logical operations . . . 45
7.4.3 Order of operations . . . 45
7.4.4 Precedence of Quantifiers . . . 45

8 Week 8 **45**
8.1 4.201 De Morgan's laws for quantifiers . . . 45
8.1.1 The intuition of De Morgan's Laws . . . 45
8.1.2 De Morgan's Laws . . . 46
8.1.3 Example 1 . . . 46
8.1.4 Negating nested quantifiers . . . 46
8.2 4.203 Rules of inference . . . 46
8.2.1 Valid argument . . . 46
8.2.2 Rules of inference . . . 47
8.2.3 Modus Ponens . . . 47
8.2.4 Modus Tollens . . . 48
8.2.5 Conjunction . . . 48
8.2.6 Simplification . . . 49
8.2.7 Addition . . . 49
8.2.8 Hypothetical Syllogism . . . 49
8.2.9 Disjunctive Syllogism . . . 50
8.2.10 Resolution . . . 50
8.2.11 Building valid arguments . . . 51

8.2.12 Fallacies . . . 52
8.3 4.205 Rules of inference with quantifiers . . . 52
8.3.1 Universal Instantiation (UI) . . . 52
8.3.2 Universal Generalisation (UG) . . . 53
8.3.3 Existential Instatiation (EI) . . . 53
8.3.4 Existential Generalization (EG) . . . 53
8.3.5 Universal Modus Ponens . . . 54
8.3.6 Universal Modus Tollens . . . 54
8.3.7 Expressing complex statements . . . 54

9 Week 9 **55**
9.1 5.101 Introduction to Boolean algebra . . . 56
9.1.1 History of Boolean Algebra . . . 56
9.1.2 Applications of Boolean Algebra . . . 56
9.1.3 Two-valued Boolean Algebra . . . 56
9.1.4 Operations of Boolean Algebra . . . 56
9.2 5.103 Postulates of Boolean algebra . . . 57
9.2.1 Huntington's postulates . . . 57
9.2.2 Basic theorems . . . 58
9.2.3 De Morgan's Theorems . . . 58
9.2.4 Principle of duality . . . 58
9.2.5 Ways of proving theorems . . . 59
9.3 5.105 Boolean functions . . . 59
9.3.1 Definition . . . 59
9.3.2 Algebraic forms . . . 60
9.3.3 Standardised forms of a function . . . 60
9.3.4 Build a *sum-of-products* form . . . 60
9.3.5 Useful functions . . . 61

10 Week 10 **61**
10.1 5.201 Logic gates . . . 61
10.1.1 Definition of a gate . . . 61
10.1.2 AND Gate . . . 61
10.1.3 OR Gate . . . 62
10.1.4 Inverter Gate . . . 62
10.1.5 XOR Gate . . . 62
10.1.6 NAND Gate . . . 63
10.1.7 NOR Gate . . . 63
10.1.8 XNOR Gate . . . 63
10.1.9 Multiple input gates . . . 63
10.1.10 Representing De Morgan's Laws . . . 64
10.2 5.203 Combinational circuits . . . 64
10.2.1 Definition of circuit . . . 64
10.2.2 Building a circuit from a function . . . 64
10.2.3 Writing Boolean expressions from a circuit . . . 66
10.2.4 Building a circuit to model a problem . . . 66
10.2.5 Building an adder circuit . . . 68
10.2.6 Building a full adder . . . 68
10.3 5.205 Simplification of circuits . . . 69
10.3.1 Benefits of Simplification . . . 69

10.3.2 Algebraic simplification 70
10.3.3 Karnaugh maps 70

11 Week 11 **71**
11.1 6.101 Introduction to proofs 71
11.1.1 Terminology 72
11.1.2 Formalising a theorem 72
11.1.3 Direct proof 72
11.1.4 Proof by contrapositive 72
11.1.5 Proof by contradiction 73
11.2 6.103 The principle of mathematical induction 73
11.2.1 The intuition behind induction 73
11.2.2 Structure of induction 74
11.2.3 Some uses of induction 74
11.3 6.106 Proof by induction 74
11.3.1 Proving formulas 74
11.3.2 Proving inequalities 75
11.3.3 Proving divisility 75
11.4 6.108 Strong induction 76
11.4.1 Strong induction 76
11.4.2 Example 76
11.4.3 Well-ordering property 77
11.4.4 Example 77
11.4.5 Equivalence of the three concepts 77

12 Week 12 **77**
12.1 6.201 Recursive definitions 78
12.1.1 Definition 78
12.1.2 Recursively defined functions 78
12.1.3 Recursively defined sets 78
12.1.4 Recursive algorithms 78
12.2 6.204 Recurrence relations 79
12.2.1 Definitions 79
12.2.2 Linear recurrences 79
12.2.3 Arithmetic sequences 79
12.2.4 Geometric sequences 79
12.2.5 Divide and conquer recurrence 79
12.3 6.206 Solving recurrence relations 80
12.3.1 Solving linear recurrence 80
12.3.2 Induction for solving recurrence 80

13 Week 13 **80**
13.1 7.101 Introduction 80
13.1.1 Applications of Graphs 81
13.2 7.103 Definition of a graph 81
13.2.1 Graph 81
13.2.2 Vertex 81
13.2.3 Edge 81
13.2.4 Adjacency 82
13.2.5 Loops and parallel edges 82

13.2.6 Directed Graph - Digraph 82
13.3 7.105 Walks and paths in a graph 83
13.3.1 Definition of a walk 83
13.3.2 Definition of a trail 83
13.3.3 Definition of a circuit 83
13.3.4 Definition of a path 84
13.3.5 Definition of a cycle 84
13.3.6 Eulerian path 84
13.3.7 Hammiltonian path 84
13.3.8 Hammiltonian cycle 84
13.3.9 Connectivity 84
13.3.10 Strong Connectivity 85
13.3.11 Transitive Closure 86
13.4 7.107 The degree sequence of a graph 86
13.4.1 Terminology - Undirected Graphs 86
13.4.2 Terminology - Directed Graphs 87
13.4.3 Degree sequence of a graph 88
13.4.4 Degree sequence property I 88
13.4.5 Degree sequence property II 88
13.5 7.109 Special graphs: simple, r-regular and complete graphs . . . 88
13.5.1 Simple Graphs 88
13.5.2 Properties of simple graphs 89
13.5.3 Regular graphs 89
13.5.4 Properties of regular graphs 89
13.5.5 Complete graph 90
13.5.6 Complete graph properties 90

14 Week 14 **90**
14.1 7.201 Isomorphic graphs 91
14.1.1 Definition of isomorphism 91
14.1.2 Properties of isomorphic graphs 91
14.2 7.203 Bipartite graphs 91
14.2.1 Matching 91
14.2.2 Maximum matching 92
14.2.3 The Hopcroft-Karp Algorithm 1 92
14.3 7.205 The adjacency matrix of a graph 92
14.3.1 Graph representation recap 92
14.3.2 Adjacency list 93
14.3.3 Example 93
14.3.4 Adjacency matrix 95
14.4 7.207 Dijkstra's algorithm 96
14.4.1 Weighted graphs 96
14.4.2 Dijstra's algorithm 97

15 Week 15 **97**
15.1 8.103 Definition of a tree 97
15.1.1 Acyclic graph 97
15.1.2 Definition of a tree 97
15.1.3 Definition of a forest 98
15.1.4 Theorem 1 98

15.1.5 Theorem 2 98
15.1.6 Rooted trees 98
15.2 8.105 Spanning trees of a graph 98
15.2.1 Constructing a spanning tree 99
15.2.2 Non iso-morphic spanning trees 100
15.3 8.107 Minimum spanning tree 100
15.3.1 Spanning tree cost 100
15.3.2 Minimum-cost spanning tree 101
15.3.3 Finding spanning trees 101

1 Week 1

Learning objectives:

- Define a set, the elements of a set and the cardinality of a set.
- Define the concepts of the universal set and the complement of a set, and the difference between a set and a powerset of a set.
- Define the concepts of the union, intersection, set difference and symmetric difference, and the concept of a membership table.

1.1 1.101 Introduction to discrete mathematics

The study of discrete objects. Such objects are separated or distant from each other.

We will study integers, propositions, sets, relations or functions.

We will learn their properties and relationships among them.

Sets, functions, logic, graphs, trees, relations, combinatorics, mathematical induction and recursive relations. We will gain mathematical understanding of these topics and that will improve our skill of thinking in abstract terms.

1.2 1.104 The definition of a set

Set Theory deals with properties of well-defined collection of objects. Introduced by George Cantor.

Forms the basis of other fields of study: counting theory, relations, graph theory and finite state machines.

1.2.1 Definition of a set

A collection of any kind of objects: people, ideas, numbers...

A set must be well-defined, meaning that there can be no ambiguity to which objects belongs to the set.

$$\begin{aligned} E &= \{2,4,6,8\} \\ V &= \{a,e,i,o,u\} \\ EmptySet &= \{\} = \emptyset \end{aligned} \tag{1}$$

Definition 1 (Set)**.** *A set is an* ***unordered*** *collection of* ***unique*** *objects.*

1.2.2 Element of a set ($\in$)

Given the set $E = \{2,4,6,8\}$ we can say $2 \in E$ (2 is an element of E) and $3 \notin E$ (3 is not an element of E)

1.2.3 Cardinality of a set (Card)

Definition 2 (Cardinality)**.** *Given a set S, the* ***cardinality*** *of S is the number of elements contained in S. We write the cardinality of S as $|S|$. Note that the cardinality of the empty set is zero ($|\emptyset| = 0$)*

1.2.4 Subset of a set ($\subseteq$)

Definition 3 (Subset)**.** *A is said to be a subset of B if and only if every element of A is also an element of B. In this case we write $A \subseteq B$.*

This means we have the following equivalence:

$$A \subseteq B \leftrightarrow \text{if x} \in A \,\text{then}\, x \in B(\text{for all x}) \tag{2}$$

The emptyset $\emptyset$ is a subset of any set.

Any set if a subset of itself ($S \subseteq S$)

1.2.5 Special Sets: $\mathbb{N}$, $\mathbb{Z}$, $\mathbb{Q}$, $\mathbb{R}$

$\mathbb{N}$: set of natural numbers

$\mathbb{Z}$: set of integers

$\mathbb{Q}$: set of rational numbers

$\mathbb{R}$: set of real numbers

1.3 1.106 The listing method and rule of inclusion

Two different ways of representing a set.

The listing method consists of simply listing all elements of a set.

$S_1 = \{1,2,3\}$

The rule of inclusion method consists of producing a rule such that when that rule is true, the element is a member of the set. For example, here's a rule of inclusion for the set of all **odd** integers:

$S_2 = \{2n + 1 \mid n \in \mathbb{Z}\}$

In some cases, the rule of inclusion (or set building notation) is the only way to actually describe a set. For example, if we were to try to list the elements of the set of rational numbers $\mathbb{Q}$, we would never be able to reach the end. However, with the set builder notation it becomes simple and concise:

$\mathbb{Q} = \{\frac{n}{m} \mid n, m \in \mathbb{Z}\text{and}m \neq 0\}$

We can use the same notation for the set of elements in my bag:

$S_{bag} = \{x \mid x\text{is in my bag}\}$

1.4 1.108 The powerset of a set

A set can contain other sets as elements. For example:

$$\begin{aligned} A &= \{1, 2, 3, 4, 5, 6, 7, 8, 9\} \\ B &= \{\{1, 2, 3, 4\}, \{5, 6\}, \{7, 8, 9\}\} \end{aligned} \tag{3}$$

Note that $\{1, 2, 3, 4\}$ is a **subset** of A but it is an **element** of B. In mathematical terms:

$$\{1, 2, 3, 4\} \subseteq A\,\text{but}\,\{1, 2, 3, 4\} \in B \tag{4}$$

1.4.1 Powerset of a set

Definition 4 (Powerset)**.** *Given a set S, the powerset of S, $P(S)$, is the set containing **all** the **subsets** of S*

Example 1 Given a set $S = \{1, 2, 3\}$, the subsets of S are:

$\emptyset, \{1\}, \{2\}, \{3\},$
$\{1, 2\}, \{1, 3\}, \{2, 3\},$
$\{1, 2, 3\}$

Therefore, the powerset of S, $P(S)$ is as follows:

$P(S) = \{\emptyset, \{1\}, \{2\}, \{3\}, \{1, 2\}, \{1, 3\}, \{2, 3\}, \{1, 2, 3\}\}$

Example 2 What is the powerset of the empty set? What is the powerset of the powerset of the empty set?

$$\begin{aligned} P(\emptyset) &= \{\emptyset\} \\ P(P(\emptyset)) &= \{\emptyset, \{\emptyset\}\} \end{aligned} \tag{5}$$

1.4.2 Cardinality of a powerset

Given a set S, then $|P(S)| = 2^{|S|}$

In other words: the cardinality of the powerset of S is the 2 to the power of the cardinality of S. For example:

$$\begin{aligned} S &= \{1, 2\} \\ |S| &= 2 \\ P(S) &= \{\emptyset, \{1\}, \{2\}, \{1, 2\}\} \\ |P(S)| &= 4 = 2^2 = 2^{|S|} \end{aligned} \tag{6}$$

Example Given a set A, if $|A| = n$ find $|P(P(P(A)))|$

$$\begin{aligned} |P(A)| &= 2^n \\ |P(P(A))| &= 2^{2^n} \\ |P(P(P(A)))| &= 2^{2^{2^n}} \end{aligned} \tag{7}$$

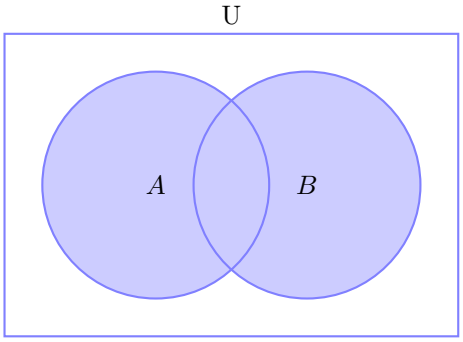
1.5 1.110 Set operations

We will look at set operations (intersection, union, difference, symmetric difference).

1.5.1 Union ($\cup$)

Definition 5 (Union)**.** *Given two sets A and B, the union of A and B, $A \cup B$, contains all the elements in **either** A or B.*

$$A \cup B = \{x \mid x \in A \text{ or } x \in B\} \tag{8}$$



Example

$$\begin{aligned} A &= \{1, 2, 3\} \\ B &= \{4, 5, 6\} \\ A \cup B &= \{1, 2, 3, 4, 5, 6\} \end{aligned} \tag{9}$$

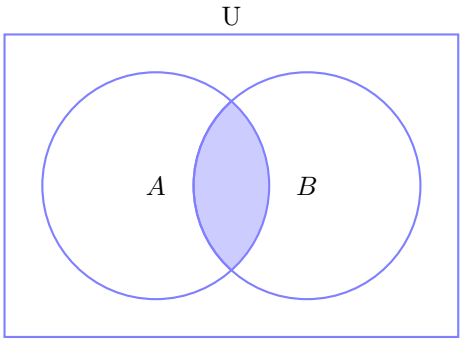
Membership Table ($A \cup B$)

A	B	$A \cup B$
0	0	0
0	1	1
1	0	1
1	1	1

1.5.2 Intersection ($\cap$)

Definition 6 (Intersection)**.** *Given two sets A and B, the intersection of A and B, $A \cap B$, contains all the elements in* ***both*** *A and B.*

$$A \cap B = \{x \mid x \in A \,\text{and}\, x \in B\} \tag{10}$$



;

Example

$$\begin{aligned} A &= \{1, 2, 3\} \\ B &= \{2, 3, 4\} \\ A \cap B &= \{2, 3, \} \end{aligned} \tag{11}$$

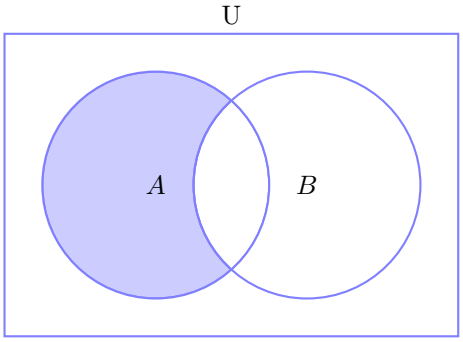
Membership Table ($A \cap B$)

A	B	$A \cap B$
0	0	0
0	1	0
1	0	0
1	1	1

1.5.3 Difference ($-$)

Definition 7 (Difference)**.** *Given two sets A and B, the difference of A and B, $A - B$, contains all the elements that are in A* ***but not*** *in B.*

$$A - B = \{x \mid x \in A \text{ and } x \notin B\} \tag{12}$$



Example

$$\begin{aligned} A &= \{1, 2, 3\} \\ B &= \{3, 4, 5\} \\ A - B &= \{1, 2, \} \end{aligned} \tag{13}$$

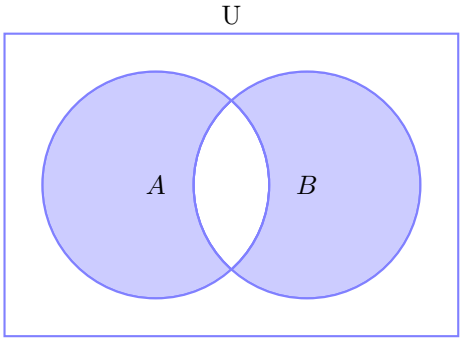
Membership Table ($A - B$)

A	B	$A - B$
0	0	0
0	1	0
1	0	1
1	1	0

1.5.4 Symmetric Difference ($\oplus$)

Definition 8 (Symmetric Difference)**.** *Given two sets A and B, the symmetric difference of A and B, $A \oplus B$, contains all the elements that are in A or in B* ***but not*** *in both.*

$$A \oplus B = \{x \mid (x \in A \text{ or } x \in B) \text{ and } x \notin A \cap B\} \tag{14}$$



Example

$$\begin{aligned} A &= \{1, 2, 3\} \\ B &= \{3, 4, 5\} \\ A \oplus B &= \{1, 2, 4, 5\} \end{aligned} \tag{15}$$

Membership Table ($A \oplus B$)

A	B	$A \oplus B$
0	0	0
0	1	1
1	0	1
1	1	0

1.5.5 Summary

Operations

$$\begin{aligned} A &= \{1, 2, 3\} \\ B &= \{3, 4, 5\} \\ A \cup B &= \{1, 2, 3, 4, 5\} \\ A \cap B &= \{3\} \\ A - B &= \{1, 2\} \\ A \oplus B &= \{1, 2, 4, 5\} \end{aligned} \tag{16}$$

Membership Table

A	B	$A \cup B$	$A \cap B$	$A - B$	$A \oplus B$
0	0	0	0	0	0
0	1	1	0	0	1
1	0	1	0	1	1
1	1	1	1	0	0

2 Week 2

Learning objectives:

- Understand the concept of Venn diagrams and how they are used to represent and compare different set expressions.
- Understand and prove De Morgan's law using membership tables.

2.1 1.201 The representation of a set using Venn diagrams

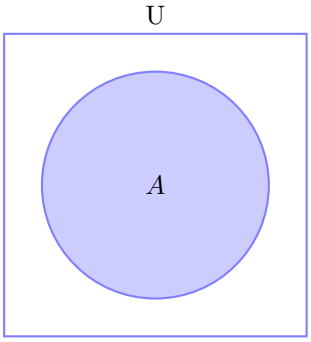
Venn diagrams can be used to represent sets and visualize the possible relations among a collection of sets. During this lesson we studied the following concepts:

- The universal set
- The complement of a set
- Set representation using Venn Diagrams

2.1.1 The Universal Set

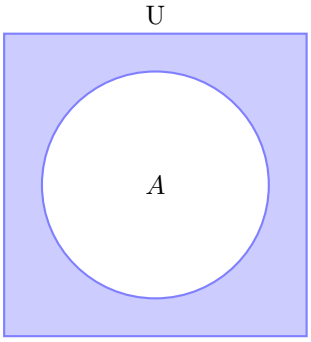
The universal set is a set containing everything. It's referred to by the letter `U`.

Note that $A \subseteq U$.



2.1.2 Complement of a set

Given a set A, the complement of A is written as $\overline{A}$, contains all the ements in the universal set U but not in A. It's represented by the area in red in figure below.



In other words $\overline{A} = U - A$.

Example

$$\begin{aligned} U &= \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10\} \\ A &= \{2, 4, 6, 8, 10\} \\ \overline{A} &= U - A \\ &= \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10\} - \{2, 4, 6, 8, 10\} \\ &= \{1, 3, 5, 7, 9\} \end{aligned} \tag{17}$$

The union of a set A with its completement $\overline{A}$ is always the universal set U.

$$A \cup \overline{A} = U \tag{18}$$

The symmetric difference of A and B is the same as the union of A and B minus the intersection of A and B:

$$A \oplus B = A \cup B - (A \cap B) \tag{19}$$

2.2 1.203 De Morgan's laws

De Morgan's laws describe how mathematical statements and concepts are related through their opposites. In se theory, they relate to intersection and unions of sets through their complements.

2.2.1 De Morgan's First Law

The complement of the union of two sets A and B is equal to the intersection of their complements.

$$\overline{A \cup B} = \overline{A} \cap \overline{B} \tag{20}$$

2.2.2 De Morgan's Second Law

The complement of the intersection of two sets A and B is equal to the union of their complements.

$$\overline{A \cap B} = \overline{A} \cup \overline{B} \tag{21}$$

2.2.3 Proof using membership tables

$\overline{A \cup B} = \overline{A} \cap \overline{B}$

A	B	$\overline{A}$	$\overline{B}$	$A \cup B$	$\overline{A \cup B}$	$\overline{A} \cap \overline{B}$
0	0	1	1	0	1	1
0	1	1	0	1	0	0
1	0	0	1	1	0	0
1	1	0	0	1	0	0

$\overline{A \cap B} = \overline{A} \cup \overline{B}$

A	B	$\overline{A}$	$\overline{B}$	$A \cap B$	$\overline{A \cap B}$	$\overline{A} \cup \overline{B}$
0	0	1	1	0	1	1
0	1	1	0	0	1	1
1	0	0	1	0	1	1
1	1	0	0	1	0	0

2.3 1.205 Laws of sets: Commutative, associative and distributive

We discussed three set identities: *Commutativity*, *Associativity*, and *Distributivity*.

2.3.1 Commutativity

When the order of operands in an operation does **NOT** affect the result, we say the operation is *commutative*. For example, addition is commutative

$$2 + 3 = 3 + 2 \tag{22}$$

Same applies for multiplication:

$$2 \cdot 3 = 3 \cdot 2 \tag{23}$$

Subtraction, however, is **NOT** commutative:

$$2 - 3 \neq 3 - 2 \tag{24}$$

In Set Theory, *Union* $\cup$, *Intersection* $\cap$, and *Symmetric Difference* $\oplus$ are all commutative operations. Much like in Algebra, Set difference is **NOT** commutative:

$$\begin{aligned} A &= \{1,2\} \\ B &= \{1,3\} \\ A - B &= \{1,2\} - \{1,3\} = \{2\} \\ B - A &= \{1,3\} - \{1,2\} = \{3\} \\ (A - B) &\neq (B - A) \end{aligned} \tag{25}$$

2.3.2 Associativity

When the grouping of elements in an operation doesn't change the result, we say the result is associative. Addition is associative:

$$(a + b) + c = a + (b + c) \tag{26}$$

In set theory, *Union*, *Intersection* and *Symmetric Difference* are all associative operations. Set difference is **not** associative:

$$\begin{aligned} A &= \{1,2\} \\ B &= \{1,3\} \\ C &= \{2,3\} \\ (A - B) - C &= (\{1,2\} - \{1,3\}) - \{2,3\} \\ &= \{2\} - \{2,3\} \\ &= \emptyset \end{aligned} \tag{27}$$

$$\begin{aligned} A - (B - C) &= \{1,2\} - (\{1,3\} - \{2,3\}) \\ &= \{1,2\} - \{1\} \\ &= \{2\} \end{aligned}$$

$$\therefore (A - B) - C \neq A - (B - C)$$

2.3.3 Distributivity

The distributive property, in general, refers to the distributive law of multiplication which states that multiplying a sum of two numbers b and c by a coefficient a is the same as multiplying each addend by the coeficient a and adding the resulting products. We say the multiplication is distributive over the addition:

$$a \cdot (b + c) = a \cdot b + a \cdot c \tag{28}$$

Similarly, the set union is distributive over set intersection:

$$A \cup (B \cap C) = (A \cup B) \cap (B \cup C) \tag{29}$$

And the set intersection is distributive over the set union:

$$A \cap (B \cup C) = (A \cap B) \cup (B \cap C) \tag{30}$$

2.3.4 Table of Set Identities

Union	Name	Intersection
$A \cup B = B \cup A$	commutative	$A \cap B = B \cap A$
$(A \cup B) \cup C = A \cup (B \cup C)$	associative	$(A \cap B) \cap C = A \cap (B \cap C)$
$A \cup (B \cap C) = (A \cup B) \cap (A \cup C)$	distributive	$A \cap (B \cup C) = (A \cap B) \cup (A \cap C)$
$\overline{A \cup B} = \overline{A} \cap \overline{B}$	De Morgan's Laws	$\overline{A \cap B} = \overline{A} \cup \overline{B}$
$A \cup \emptyset = A$	identities	$A \cap \emptyset = \emptyset$
$A \cup U = U$		$A \cap U = A$
$A \cup \overline{A} = U$	complement	$A \cap \overline{A} = \emptyset$
$\overline{U} = \emptyset$		$\overline{\emptyset} = U$
$\overline{\overline{A}} = A$	double complement	
$A \cup (A \cap B) = A$	absorption	$A \cap (A \cup B) = A$
$A - B = A \cap \overline{B}$	set difference	

2.3.5 Applying set identities to simplify expressions

Show that $\overline{(A \cap B) \cup \overline{B}} = B \cap \overline{A}$

$$\begin{aligned}
\overline{(A \cap B) \cup \overline{B}} &= \overline{(A \cap B)} \cap \overline{\overline{B}} \\
&= \overline{(A \cap B)} \cap B \\
&= (\overline{A} \cup \overline{B}) \cap B \\
&= \overline{A} \cap B \cup \overline{B} \cap B \\
&= \overline{A} \cap B \cup \emptyset \\
&= \overline{A} \cap B \\
&= B \cap \overline{A}
\end{aligned} \tag{31}$$

2.4 1.207 Partition

A partition of an object is a subdivision of the object into parts such that the parts are completely separated from each other, yet together they form the whole object.

Data partitioning has many applications in Computer Science such as Big Data analysis. This is usually referred to as *Divide and Conquer* approach. Such techniques must be applied in cases where the entire input data doesn't fit into

the physical memory of the Computer. In such cases, we must find a way to partition the data so that subsets of the original data can be operated on without changing the result of the whole computation.

2.4.1 Definition of a partition of a set

Two sets A and B are said to be disjointed if and only if $A \cap B = \emptyset$.
Definition 9 (Set Partition)**.** *A partition of set A is a set of subsets A_i such that all subsets are disjointed and then union of all subsets A_i is equal to A.*

3 Week 3

Learning objectives:

- Define a function.
- Describe the properties of functions.
- Explain how to plot a function.

3.1 2.101 Introduction

A function is a rule that relates to how one quantity depends on another quantity. Much like a voltage depends on electrical current and resistance.

During this lecture, we learn the definition of a function and study a few of their properties.

3.2 2.102 The Definition of A Function

A function is a relation between a set of inputs and a set of outputs such that each input maps to exactly **one** output.

3.2.1 Definition

A function maps an element of set 1 to an element in set 2. Such mapping is *well-behaved* meaning that given a starting point we always know exactly where to go. For example, we could have a function that maps a set of strings to their corresponding number of characters:

$$\begin{aligned} S_1 &= \{Sea, Land, Sky\} \\ S_2 &= \{1, 2, 3, 4, 5, 6\} \end{aligned} \tag{32}$$

$$\begin{aligned} Sea &\to 3 \\ Land &\to 4 \\ Sky &\to 3 \end{aligned}$$

From Rosen's book, functions are defined as:
Definition 10 (Function)**.** *Let A and B be nonempty sets. A function f from A to B is an assignment of exactly one element of B to each element of A. We write $f(a) = b$ if b is the unique element of B assigned by the function f to the element a of A. If f is a function from A to B, we write $f : A \to B$ and read as f maps A to B.*

$x \in A : x \to f(x) = y \, (y \in B)$

3.2.2 Domain, co-domain and range of a function

Given a function $f : A \to B$

$x \in A \to f(x) = y \in B$

A is the set of inputs and its referred to as the *Domain of f.* We write it as $D_f = A$.

B is the set containing all possible outputs; referred to as the *co-domain of f.* We write it as $co - D_f = B$.

The set containing all outputs is called the *Range of f* and is written as R_f.

3.2.3 Image and pre-image (antecedent) of an element

y, the output of the function of a given input x, is called the *Image of* x where x itself is called the *pre-image of y.* We write $f(x) = y$.

3.2.4 Example of Domain, co-domain and range

Let A be the set $\{On, Sea, Land, Sky\}$, B be the set $\{1, 2, 3, 4, 5, 6\}$, and f be the function that maps the set of strings to their corresponding number of characters. We have:

$$\begin{aligned} On &\to 2 \\ Sea &\to 3 \\ Land &\to 4 \\ Sky &\to 3 \end{aligned} \tag{33}$$

In this case:

$$\begin{aligned} D_f = A &= \{On, Sea, Land, Sky\} \\ co - D_f = B &= \{1, 2, 3, 4, 5, 6\} \\ R_f &= \{2, 3, 4\} \end{aligned} \tag{34}$$

Moreover, we can say that 2 is the image of the string On and On is the pre-image of 2. $Pre - images(2) = \{On\}$.

3 is the image of Sea and Sky, therefore $Pre - images(3) = \{Sea, Sky\}$.

3.3 2.104 Plotting functions

We explore and plot some special functions.

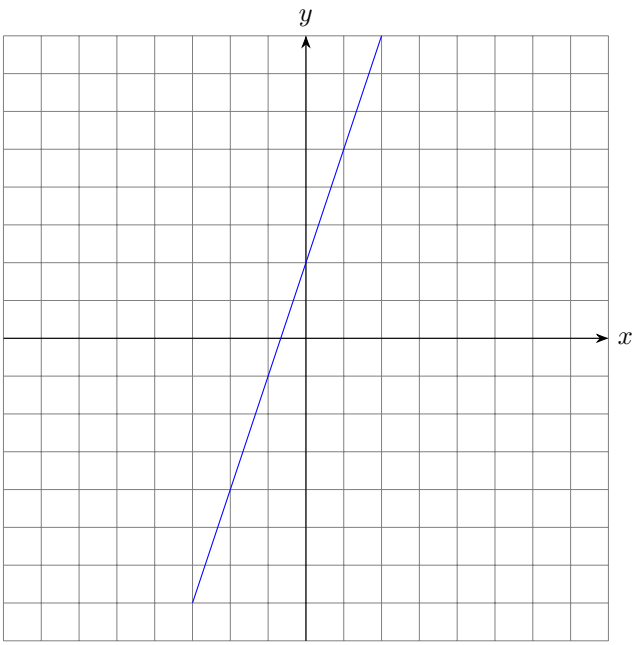
3.3.1 Linear Functions

A function f is called a linear function if it is of the form $f(x) = ax + b$. This function is a straight line passing through the point $(0, b)$ with gradient a.

If $a > 0$, then the function is increasing. It's decreasing if $a < 0$.

In order to plot this function, first we make a table of values for this function. We use $f(x) = 3x + 2$ as an example.

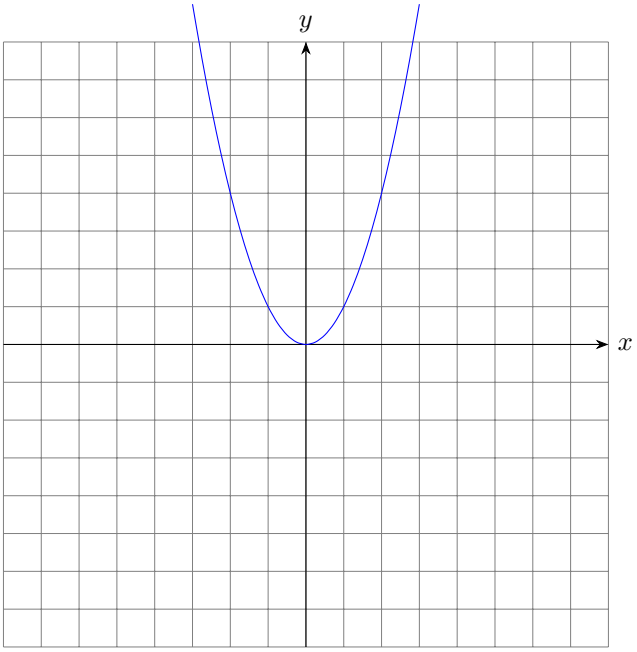
x	f(x)
0	2
1	5
2	8
3	11
4	14



3.3.2 Quadratic functions

A function f of the form $f(x) = ax^2 + bx + c$ is called a *Quadratic function.*

x	f(x)
0	0
1	1
2	4
3	9
4	16

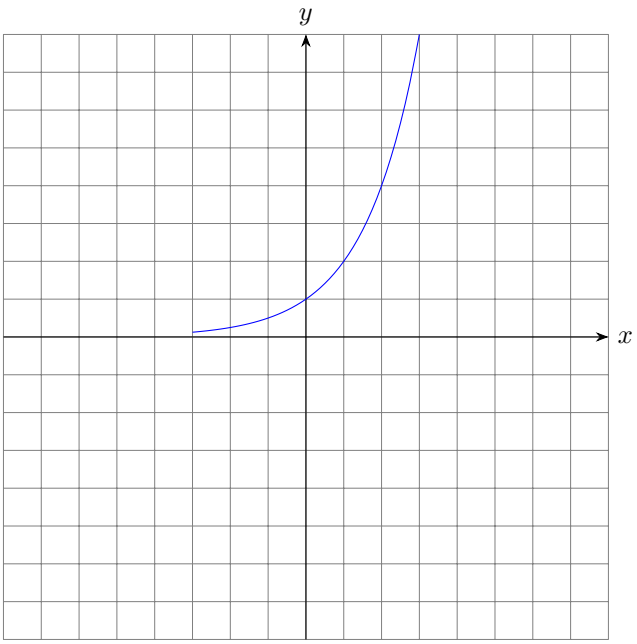


3.3.3 Exponential functions

A function f of the form $f(X) = b^x$ is called an *exponential function*. The variable b is called the *base* of the function.

A more formal definition may be:
Definition 11 (Exponential Function)**.** *The function f defined by $f : \mathbb{R} \to \mathbb{R}^+$ and $f(x) = b^x$ where $b > 0$ and $b \neq 1$ is called and exponential function with a base b.*

x	f(x)
0	1
1	2
2	4
3	8
4	16



Exponentials have some properties which are good to remember:

Form	Result
$b^x \cdot b^y$	b^{x+y}
$\frac{b^x}{b^y}$	b^{x-y}
$(b^x)^y$	$b^{x \cdot y}$
$(a \cdot b)^x$	$a^x \cdot b^x$
$\left(\frac{a}{b}\right)^x$	$\frac{a^x}{b^x}$
b^{-x}	$\frac{1}{b^x}$

The point $(0, 1)$ is the common point for all exponentials. When $b > 1$ we have an exponential growth. When $0 < b < 1$, we have exponential decay.

3.4 2.106 Injective and surjective functions

3.4.1 Injective Functions

Let $f : A \to B$ be a function; f is said to be injective, or *one-to-one* if and only if $\forall a, b \in A$, if $a \neq b$ then $f(a) \neq f(b)$. In plain english, this means that two different inputs will lead to two different outputs, i.e. given two different inputs a and b, then the **image** of a is different than the image of b.

A corollary of this is that:
Corollary 1. $\forall a, b \in A, f(a) = f(b) \to a = b$

Example: linear function Show that a function $f : \mathbb{R} \to \mathbb{R}$ with $f(x) = 2x + 3$ is an injection (one-to-one).

We can prove this in two different ways. The first proof assumes $f(a) = f(b)$

Proof. Let $a, b \in \mathbb{R}$, show that if $f(a) = f(b)$ then $a = b$.

$$\begin{aligned} f(a) = f(b) \rightarrow & 2a + 3 = 2b + 3 \\ & 2a + 3 - 3 = 2b + 3 - 3 \\ & 2a = 2b \\ & \frac{2a}{2} = \frac{2b}{2} \\ & a = b \end{aligned} \tag{35}$$

$\therefore f$ is injective. □

The second proof assumes $a \neq b$

Proof. Let $a, b \in \mathbb{R}$, show that if $a \neq b$ then $f(a) \neq f(b)$.

$$\begin{aligned} a \neq b \rightarrow & 2a \neq 2b \\ & 2a + 3 \neq 2b + 3 \\ & f(a) \neq f(b) \end{aligned} \tag{36}$$

$\therefore f$ is injective. □

Example: quadratic function To prove that a function is not injective, we only need to find one example of two different inputs having the same image.

Show that a function $f : \mathbb{R} \rightarrow \mathbb{R}$ with $f(x) = x^2$ is not injective.

Proof.

$$\begin{aligned} & f(5) = (5)^2 = (-5)^2 = f(-5) \\ & \text{however } 5 \neq -5 \end{aligned} \tag{37}$$

$\therefore f$ is not injective. □

However, if we change the domain of the function such that $f : \mathbb{R}^+ \rightarrow \mathbb{R}$, we can make it injective. To prove this, we can apply the same two methodologies from the previous example.

3.4.2 Surjective Functions

Let $f : A \rightarrow B$ be a function; f is said to be surjective, or *onto* if and only if $\forall y \in B \, \exists x \in A \mid y = f(x)$. This means that every element in the co-domain of f, B, has **at least** one pre-image in the domain of f, A. This is equivalent to saying that the range and the co-domain of a surjective function, are equal (i.e. $R_f = co - D_f$).

Example: linear function Show that a function $f : \mathbb{R} \to \mathbb{R}$ with $f(x) = 2x + 3$ is a surjection (onto).

To prove this, we must show that for every element in B, there is a pre-image in A.

Proof. Let $y \in \mathbb{R}$, show that $\exists x \in \mathbb{R} \mid f(x) = y$.

$$\begin{aligned} f(x) = y \rightarrow & 2x + 3 = y \\ & 2x + 3 - 3 = y - 3 \\ & \frac{2x}{2} = \frac{y-3}{2} \\ & x = \frac{y-3}{2} \in \mathbb{R} \end{aligned} \tag{38}$$

$\therefore \forall y \in \mathbb{R} \exists x = \frac{y-3}{2} \in \mathbb{R} \mid f(x) = y$, hence f is surjective. □

Example: quadratic function Show that a function $f : \mathbb{R} \to \mathbb{R}$ with $f(x) = x^2$ is not a surjection.

Proof. Let $y \in \mathbb{R}$, show that $\exists x \in \mathbb{R} \mid f(x) = y$.

Let $y \in \mathbb{R}$, show that $\exists x \in \mathbb{R} \mid f(x) = y$.

$R_f = [0, +\infty[\neq co - D_f = \mathbb{R}$

$\therefore f$ is not surjective. □

4 Week 4

Learning objectives:

- Discuss special functions.
- Describe inverse functions.

4.1 2.201 Function composition

Using examples we will understand function composition and how to work out the composition of two functions. We will also show that function composition is **not** commutative.

Given two functions, f and g, the composition of f and g is written as $f \circ g = f(g(x))$.

For example, let $f(x) = 2x$ and $g(x) = x^2$, the composition of f and g can be worked out as follows:

$$\begin{aligned}(f \circ g)(x) &= f(g(x)) \\ &= f(x^2) \\ &= 2x^2\end{aligned}$$

$$\begin{aligned}(f \circ g)(1) &= f(g(x)) \\ &= f(1^2) \\ &= 2 \cdot 1^2 \\ &= 2\end{aligned}$$

What this means is that if we have a function $g : A \to B$ and a function $f : B \to C$, function composition allows us to produce a function $(f \circ g) : A \to C$.

Note that function composition is **not** commutative. In other words, $f \circ g \neq g \circ f$. Let $f = 2x$ and $g = x^2$, we can show that $(f \circ g) = 2x^2$ and $(g \circ f) = 4x^2$.

4.2 2.203 Bijective functions

4.2.1 Definition

A bijective or invertible function is a function $f : A \to B$ that can be described as both *injective* and *surjective* simultaneously. This means that each element of the *co-domain* has exactly one *pre-image.*

Definition 12 (Bijection)**.** *A function $f(x)$ is said to be bijective if and only if it is both injective and surjective.*

4.2.2 Exercise 1:

Show that the function $f : \mathbb{R} \to \mathbb{R}$ with $f(x) = 2x + 3$ is a bijective (invertible) function.

Proof. To prove this, suffices to prove that this function is both an injection and a surjection. Let's prove the injection case first:

Let $a, b \in \mathbb{R}$, we will show that if $f(a) = f(b)$ then $a = b$.

$$\begin{aligned}f(a) = f(b) \to & 2a + 3 = 2b + 3 \\ & 2a + 3 - 3 = 2b + 3 - 3 \\ & 2a = 2b \\ & \frac{2a}{2} = \frac{2b}{2} \\ & a = b\end{aligned}$$

$\therefore f$ is injective.

Now turning our attention to the surjection case, we have:

Let $y \in \mathbb{R}$, we will show that $\forall y \in \mathbb{R} \exists x \in \mathbb{R} \mid f(x) = y$.

$$\begin{aligned} f(x) = y \rightarrow & 2x + 3 = y \\ & 2x + 3 - 3 = y - 3 \\ & \frac{2x}{2} = \frac{y-3}{2} \\ & x = \frac{y-3}{2} \in \mathbb{R} \end{aligned}$$

$\therefore \forall y \in \mathbb{R} \exists x = \frac{y-3}{2} \in \mathbb{R} \mid f(x) = y$, hence f is surjective.

Because we have proved that $f(x) = 2x+3$ is both an injection and a surjection, we have also proved that it is a bijection. □

4.2.3 Inverse function

Definition 13 (Inverse function)**.** *Let $f : A \rightarrow B$, if f is bijective, then the inverse function f^{-1} exists and is defined as $f^{-1} : B \rightarrow A$.*

Given this definition, let's find the inverse of $2x + 3$.

4.2.4 Exercise 2:

The following function $f : \mathbb{R} \rightarrow \mathbb{R}$ with $f(x) = 2x + 3$ is a bijection. Find the inverse function f^{-1}.

$$\begin{aligned} f(x) &= 2x + 3 \\ f(x) &= y \\ 2x + 3 &= y \\ 2x + 3 - 3 &= y - 3 \\ \frac{2x}{2} &= \frac{y-3}{2} \\ x &= \frac{y-3}{2} \end{aligned}$$

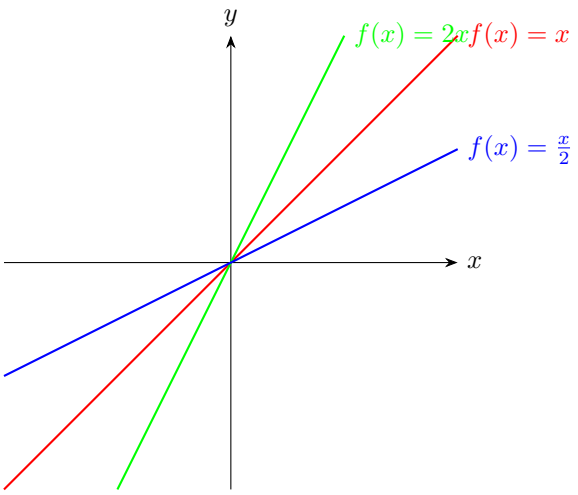
$$\therefore f^{-1}(x) = \frac{x-3}{2}$$

4.2.5 Identity function

There is one special case of composition which is $(f \circ f^{-1})(x) = (f^{-1} \circ f)(x) = x$. For example if $f(x) = 2x$, then $f^{-1}(x) = \frac{x}{2}$, therefore $(f \circ f^{-1})(x) = 2\frac{x}{2} = x$. Similarly, $(f^{-1} \circ f)(x) = \frac{2x}{2} = x$.

4.2.6 Plotting the inverse function

The function f and its inverse f^{-1} are always symmetric to the straight line $y = x$.



4.3 2.205 Logarithmic functions

Exponential and logarithmic functions are closely related. Therfore, let's review exponential functions before dealing with logarithmic functions.

Exponential functions were defined back in Definition 11. We know from that definition that:

$$y = f(x) = b^x \ (b > 0, \ b \neg 1)$$

The domain of the function is $(-\infty, +\infty)$.

The range of the function is $(0, +\infty)$.

The graph of an exponential function **always** passes through the point with coordinates $(0, 1)$. If the base b is greater than 1, then the function is increasing on $(-\infty, +\infty)$ and we call it *exponential growth*. Conversely, if $b < 1$, then the function is decreasing on $(-\infty, +\infty)$ and we call it *exponential decay*.

4.3.1 Definition

With that review out of the way, we can define Logarithmic functions:
Definition 14 (Logarithmic function)**.** *The logarithmic function with base b where $b > 0$ and $b \neq 1$ is defined as follows:*

$$log_b x = y \leftrightarrow x = b^y$$

We can say that $log_b x$ is the inverse function of the exponential function b^x.

4.3.2 Laws of logarithmic functions

1. $log_b m \times n = log_b m + log_b n$
2. $log_b \frac{m}{n} = log_b m - log_b n$
3. $log_b m^n = n\ timeslog_b m$
4. $log_b 1 = 0$
5. $log_b b = 1$

4.3.3 Exercise 1

$log_3 81 \quad log_3 81 = log_3 3^4 = 4 \times log_3 3 = 4 \times 1 = 1$

$log_{10} 100 \quad log_{10} 100 = log_{10} 10^2 = 2 \times log_{10} 10 = 2 \times 1 = 2$

$log_3 \frac{1}{81} \quad log_3 \frac{1}{81} = log_3 81^{-1} = log_3 3^{-4} = -4 \times log_3 3 = -4 \times 1 = -4$

$log_2 1 \quad log_2 1 = log_2 2^0 = 0 \times log_2 2 = 0 \times 1 = 0$

4.3.4 Natural logarithm

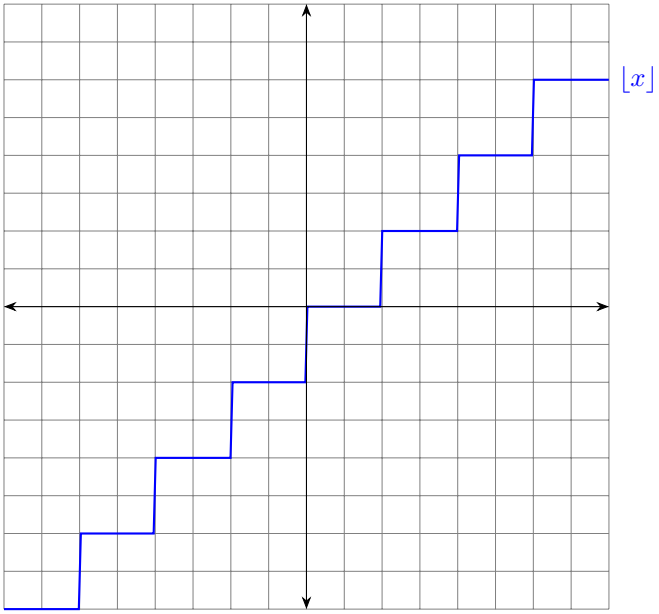
The natural logarithm, commonly written as $ln(x)$ is the logarithm with base e. In other words: $ln(x) = log_e x$ where $e \approx 2.71828$.

4.4 2.207 Floor and ceiling functions

4.4.1 Floor function

Definition 15 (Floor function)**.** *The **floor** function is a function $f : \mathbb{R} \to \mathbb{Z}$. It takes a real number x as input and outputs the largest integer that is less than or equal to x. Denoted as $floor(x) = \lfloor x \rfloor$.*

For example, given a real number x such that $n \leq x < n + 1$, the floor of x is n. In other words: $floor(x) = \lfloor x \rfloor = n$.

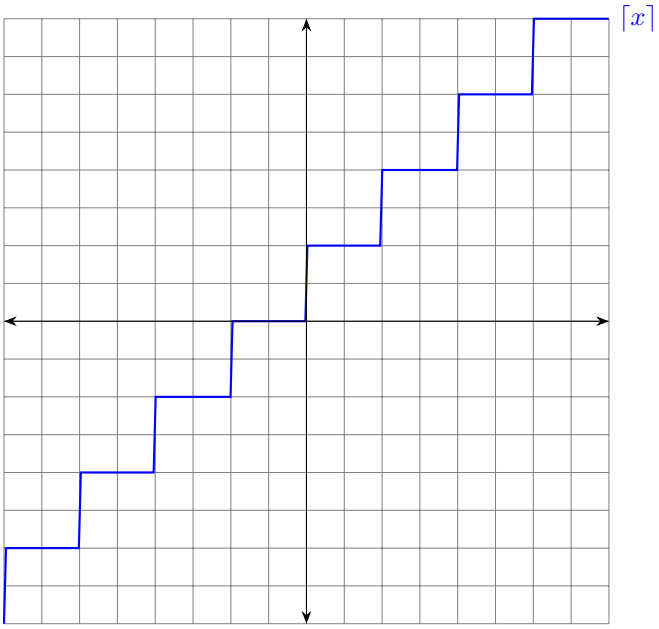


We can think of the floor function as if we're walking on the number line to the left until we find an integer. This means that $\lfloor 1.1 \rfloor = 1$ but $\lfloor -1.1 \rfloor = -2$.

4.4.2 Ceiling function

Definition 16 (Ceiling function)**.** *The **ceiling** function is a function $f : \mathbb{R} \to \mathbb{Z}$. It takes a real number x as input and outputs the smallest integer that is greater than or equal to x. Denoted as $ceiling(x) = \lceil x \rceil$.*

For example, given a real number x such that $n < x \leq n + 1$, the ceiling of x is $n + 1$. In other words: $ceiling(x) = \lceil x \rceil = n + 1$.



This is exact opposite of the floor function. So we can think of it as if were were walking on the number line to the right until we find an integer. This means that $\lfloor 1.1 \rfloor = 2$, but $\lfloor -1.1 \rfloor = -1$.

4.4.3 Exercise 1

Let n be an integer and x a real number. Show that:

$$\lfloor x + n \rfloor = \lfloor x \rfloor + n$$

Proof. Let m be an integer such that $m = \lfloor x \rfloor$. By definition of the floor function we have $m \leq x < m + 1$. Addin n to both sides of this inequality, we have $m + n \leq x + n < m + n + 1$.

This implies that $\lfloor x + n \rfloor = m + n$ by definition. And $m = \lfloor x \rfloor$. Therefore $\lfloor x + n \rfloor = \lfloor x \rfloor + n$. □

5 Week 5

Learning Objectives

- Explain and apply basic concepts of propositional logic.
- Construct truth tables of propositions and use them to demonstrate the equivalence of logical statements.
- Translate natural language statements into symbolic logical statements and vice versa.

5.1 3.101 Introduction to propositional logic

Definition 17 (Propositional Logic)**.** *It is a branch of logic that is interested in studying mathematical statements.*

Propositional Logic is the basis of all mathematical reasoning and the rules used to construct mathematical theories. Its original purpose was to model reasoning and dates back to Aristotle.

Effectively, it is an *algebra of propositions*. In this *algebra*, the variables are unknown **propositions** rather than unknown **real numbers**.

The operators used are and ($\wedge$), or ($\vee$), not ($\neg$), implies ($\rightarrow$) and if and only if ($\leftrightarrow$) instead of our regular $+$, $-$, $\times$, and $\div$.

5.1.1 Applications of propositional logic

Propositional logic can be used in logic circuit design. It can also be applied to programming languages, such as Prolog.

Many computer reasoning systems, including theorem provers, program verifiers and applications in the field of Artificial Intelligence, have been implemented in logic-based programming languages.

These languages, generally employ predicate logic, a form of logic that extends the capabilities of propositional logic.

5.2 3.103 Propositions

Definition 18 (Proposition)**.** *A declarative sentence that is either true or false, but not both.*

A Proposition is the most basic element of logic. Which means that propositions are the building blocks for our reasoning and logical statements.

5.2.1 Examples of propositions

As mentioned above, a proposition must be a declarative statement that is either true or false, therefore the following statements are propositions:

- **London is the capital of the United Kingdom**

  We know this is true, so this is considered to be a **true proposition**.

- $\mathbf{1 + 1 = 2}$

  This is also a **true proposition**.

- $\mathbf{2 < 3}$

  This is also a **true proposition**.

- **Madrid is the capital of France**

  This is a **false proposition**.

- **$3 < 2$**

  This is also a **false proposition**.

- **10 is an odd number**

  This is also a **false proposition**.

What follows is a series of statements which are **not** propositions, as they can not assume a true or false value:

- $x + 1 = 2$

  We don't know the value of x, so this is **not** a proposition. However, if a value is assigned to x, then at that moment it becomes a proposition. IF we assign the value 1 to x, this will be a **true** proposition, if any other value is assigned to x, then it'll be a **false** proposition.

- $x + y = z$

  Also not a proposition as x, y and z have no values.

- What time is it?

  Is not a proposition as it is not a declarative sentence.

5.2.2 Propositional Variables

To avoid writing long, repetivive propositions, we make use of **propositional variables**. They are typically a letter, such as **p**, **q**, **r**, ...

We can assign letters to our previous propositions, for example: Let p be the proposition *London is the capital of the United Kingdom.*

5.3 3.105 Truth tables and truth sets

As we begin to build more complex compound propositions, we need a method of keeping track of this proposition's truth value. A truth table is one such method.

5.3.1 True Tables

A truth table is tabular representation of all the possible combinations of truth values for a set of propositional variables.

For example:

p	q
F	F
F	T
T	F
T	T

A truth table of n propositional variables, will contain 2^n rows. So a table for 3 propositional variables, will have 8 rows:

p	q	r
F	F	F
F	F	T
F	T	F
F	T	T
T	F	F
T	F	T
T	T	F
T	T	T

5.3.2 Truth Set

Definition 19 (Truth Set)**.** *Let p be a proposition on a set S. The truth set of p is the set of elements of S for which p is true.*

Commonly, we use a capital letter to refer to the truth set of a proposition. For example the truth set of a proposition p is referred to as P.

Example Let $S = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10\}$.

Let p and q be two propositions concerning an integer n in S, defined as follows:

$p : n$ is even
$q : n$ is odd

Therefore, the truth set of p is $P = \{2, 4, 6, 8, 10\}$ and the truth set of q is $Q = \{1, 3, 5, 7, 9\}$.

5.4 3.107 Compound propositions

Compound propositions are built by combining propositions with logical operators (also referred to as connectives). The connectives which we deal with in this lecture are:

- **Negation** $\neg$
- **Conjunction** $\wedge$
- **Disjunction** $\vee$
- **Exclusive-or** $\oplus$

5.4.1 Negation $\neg$

Let p be a proposition, the negation of p, denoted by $\neg p$, and read as "not p", is the statement: *it is **not** the case that p.*

For example if p is the statement *John's program is written in Python*, then $\neg p$ is the statement *John's program is **not** written in Python.*

p	$\neg p$
F	T
T	F

5.4.2 Conjunction $\wedge$

Let p and q be propositions, the conjuntion of p and q, denoted by $p \wedge q$, and read as "p and q", is the statement: *p and q.*

The conjunction is only true when both p and q are true and false otherwise.

For example if p is the statement *John's program is written in Python*, and q is the statement *John's program has less then 20 lines of code*, then $p \wedge q$ is the statement *John's program is written in Python **and** has less than 20 lines of code.*

p	q	$p \wedge q$
F	F	F
F	T	F
T	F	F
T	T	T

5.4.3 Disjunction $\vee$

Let p and q be propositions, the disjuntion of p and q, denoted by $p \vee q$, and read as "p or q", is the statement: *p or q.*

The disjunction is only false when both p and q are false and true otherwise.

For example if p is the statement *John's program is written in Python*, and q is the statement *John's program has less then 20 lines of code*, then $p \vee q$ is the statement *John's program is written in Python **or** has less than 20 lines of code.*

p	q	$p \vee q$
F	F	F
F	T	T
T	F	T
T	T	T

5.4.4 Exclusive-or $\oplus$

Let p and q be propositions, the exclusive-or of p and q, denoted by $p \oplus q$, and read as "p exclusive-or q", is the statement: *p exclusive-or q.*

The exclusive is true when either p or q are true, but not both.

For example if p is the statement *John's program is written in Python*, and q is the statement *John's program has less then 20 lines of code*, then $p \oplus q$ is the statement *John's program is written in Python* ***or*** *has less than 20 lines of code,* ***but not both****.*

p	q	$p \oplus q$
F	F	F
F	T	T
T	F	T
T	T	F

5.4.5 Precedence of logical operators

Propositions can be combined to build complex compound propositions. To do this we need to start relying on precedence of logical operators or use parenthesis.

The meaning of compound propositions can change depending on the order in which parentheses are used. For example $(p \vee q) \wedge (\neg r) \neq p \vee (q \wedge \neg r)$.

Here's a small table of precedence:

Operator	Precedence
$\neg$	1
$\wedge$	2
$\vee$	3

5.4.6 Exercise

Given a positive integer n, let's consider the propositions p and q, where:

- p: *n is an even number*
- q: *n is less than 10*

Let's write the logical expression for each of the following propositions:

1. *n is an even number* **and** *is less than 10* $p \wedge q$
2. *n is either an even number* **or** *is less than 10* $p \vee q$
3. *n is either an even number* **or** *is less than 10* **but not both** $p \oplus q$
4. $\neg p \vee (p \wedge q)$

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

6 Week 6

Learning Objectives

- How to formalise a logical implication
- Apply the laws of propositional to analyse propositions and arguments.

6.1 3.202 Logical implication ($\rightarrow$)

Definition 20 (Implication)**.** *Let p and q be propositions. The conditional statement or implication $p \rightarrow q$ is the proposition "if p then q".*

p is called the hypothesis (or antecedent) q is called the conclusion (or consequence)

Example 1 Let p and q be the following statements:

- p: John did well in Discrete Mathematics
- q: John will do well in the Programming Course

The conditional statement $p \rightarrow q$ can be written as follows:

If John did well in Discrete Mathematics then John will do well in the Programming Course.

6.1.1 Truth Table

p	q	$p \rightarrow q$
F	F	T
F	T	T
T	F	F
T	T	T

As we can see, the only situation where an implication evaluates to false is when our hypothesis is true but the conclusion is false.

6.1.2 Different expressions for $p \rightarrow q$

Let p and q be the following statements:

- p: It's sunny
- q: John goes to the park

There are many ways to write the conditional statement $p \rightarrow q$:

- $p \rightarrow q$
- if p **then** q
- if p, q

- p implies q
- p only if q
- q follows from p
- p is sufficient for q
- q unless $\neg p$
- q is necessary for p

All of these forms are equivalent to *if it's sunny then John goes to the park.*

Note that the statement *John going to the park is necessary for a sunny day* sounds a bit strange in English. We should try to think of it as *John going to the park is a necessary consequence of a sunny day.*

6.1.3 Converse, inverse and contrapositive

Let p and q be propositions and A the conditional statement $p \to q$.

The conditional statement $q \to p$ is referred to as the **converse** of A.

The conditional statement $\neg q \to \neg p$ is referred to as the **contrapositive** of A.

The conditional statement $\neg p \to \neg q$ is referred to as the **inverse** of A.

The **contrapositive** of A has the same truth table as A and is, therefore, equivalent to it.

Example 2 Let p and q be the following statements:

- p: It's sunny
- q: John goes to the park
- $A = p \to q$: If it's sunny then John goes to the park

Therefore:

- **converse**: If John goes to the park, then it's sunny
- **contrapositive**: If John does not go to the park, then it's not sunny
- **inverse**: If it's not sunny then John does not go the park

We can build a large truth table with all of these:

p	q	$p \to q$	$\neg q \to \neg p$	$\neg p \to \neg q$	$q \to p$
F	F	T	T	T	T
F	T	T	T	F	F
T	F	F	F	T	T
T	T	T	T	T	T

Note, also, that the **converse** or A and the **inverse** of A are equivalent.

6.2 3.204 Logical equivalence ($\leftrightarrow$)

Definition 21 (Logical Equivalence)**.** *Let p and q be propositions. The **biconditional** or **equivalence** statement $p \leftrightarrow q$ is the proposition $p \to q \wedge q \to p$.*

Biconditional statements are also called bi-implications and can be read p ***if and only if*** q

6.2.1 Truth Table

p	q	$p \leftrightarrow q$
F	F	T
F	T	F
T	F	F
T	T	T

We can see here that the biconditional statement of p and q is true whenever p and q have the same truth value and is false otherwise.

6.2.2 Equivalent propositions

Let p and q be propositions. We say that p and q are *logically equivalent* if they always have the same truth value.

We write $p \equiv q$ to signify that p is equivalent to q.

Note that $\equiv$ is not a logical operator, and $p \equiv q$ is not a compound proposition. $p \equiv q$ means that the compound proposition $p \leftrightarrow q$ is always true.

6.2.3 Proving equivalence

One way of determining logical equivalence, is by means of truth tables and verifying that two propositions have the same truth values for every possible input.

p	q	$p \to q$	$\neg p$	$\neg p \vee q$
F	F	T	T	T
F	T	T	T	T
T	F	F	F	F
T	T	T	F	T

If values difer in any row, then we demonstrate non-equivalence.

Example 1 Let p, q, and r be the following propositions concerning n:

- p: $n = 20$
- q: n is even
- r: n is positive

Let's express each conditional statement below symbolically:

- **If** $n = 20$, **then** n is positive

  $p \to r$

- $n = 20$ **if** n is even

  $q \to p$

- $n = 20$ **only if** n is even

  $p \to q$

6.2.4 Precedence of logical operators (Updated)

Propositions can be combined to build complex compound propositions. To do this we need to start relying on precedence of logical operators or use parenthesis.

The meaning of compound propositions can change depending on the order in which parentheses are used. For example $(p \vee q) \wedge (\neg r) \neq p \vee (q \wedge \neg r)$.

Here's a small table of precedence:

Operator	Name	Precedence
$\neg$	Negation	1
$\wedge$	Conjunction	2
$\vee$	Disjunction	3
$\to$	Conditional or Implication	4
$\leftrightarrow$	Biconditional or Equivalence	5

6.3 3.206 Laws of propositional logic

The following table summarizes the laws of Propositional Logic.

	Disjunction	Conjunction
Idempotent laws	$p \vee p \equiv p$	$p \wedge p \equiv p$
Commutative laws	$p \vee q \equiv q \vee p$	$p \wedge q \equiv q \wedge p$
Associative laws	$(p \vee q) \vee r \equiv p \vee (q \vee r)$	$(p \wedge q) \wedge r \equiv p \wedge (q \wedge r)$
Distributive laws	$p \vee (q \wedge r) \equiv (p \vee q) \wedge (p \vee r)$	$p \wedge (q \vee r) \equiv (p \wedge q) \vee (p \wedge r)$
Identity laws	$p \vee \mathbf{F} \equiv p$	$p \wedge \mathbf{T} \equiv p$
Domination laws	$p \vee \mathbf{T} \equiv \mathbf{T}$	$p \wedge \mathbf{F} \equiv \mathbf{F}$
De Morgan's laws	$\neg(p \vee q) \equiv \neg p \wedge \neg q$	$\neg(p \wedge q) \equiv \neg p \vee \neg q$
Absorption laws	$p \vee (p \wedge q) \equiv p$	$p \wedge (p \vee q) \equiv p$
Negation laws	$p \vee \neg p \equiv \mathbf{T}$[1]	$p \wedge \neg p \equiv \mathbf{F}$[2]
Double negation law	$\neg\neg p \equiv p$	

[1] A statement that's always **true** is a *Tautology*
[2] A statement that's always **false** is a *Contradiction*

7 Week 7

Learning Objectives

- Describe the basic concepts of predicate logic.
- Describe existential and universal quantifiers.
- Assign truth values to quantified statements.

7.1 4.101 Introduction to predicate logic

Our previous Propositional Logic is useful for studying propositions but has some limitations:

- It cannot express precisely the meaning of complex mathematical statements.
- It only studies propositions, i.e. statements with known truth values

Predicate Logic is a different type of Mathematical Logic which overcomes the limitations of Propositional Logic and can be used to build more complex reasoning.

7.1.1 Example 1

Given the statements:

- *All* ***men*** *are* ***mortal****.*
- ***Socrates*** *is a* ***man****.*

It's natural to conclude that *Socrates* is a *man*. This sort of reasoning cannot be expressed by Propositional Logic. Predicate Logic enables us to formalise it.

7.1.2 Example 2

Given the statement ***x*** *square is equal to* ***4***. We **know** this statement is **NOT** a proposition as its truth value is a function depending on **x**, however Predicate Logic can express and formalise this statement.

7.2 4.103 What are predicates?

We start with some examples of statements which cannot be expressed by Propositional Logic.

7.2.1 Insufficiency of Propositional Logic

Going back to the previous example ***x squared is equal to 4***. We already saw that this statement cannot be a proposition because its truth value is a function depending on **x**.

7.2.2 Definition of Predicate

Predicates are generalizations of propositions. They are (Boolean) functions which return *TRUE* or *FALSE* depending on their variables. They **become** propositions when their variables are assigned values.

Predicates, much like regular sentences, are composed of smaller parts. The statement ***x square is equal to 4*** contains two parts: the variable **x**; and the predicate **is equal to 4**.

We can formalize this statement as $P(x)$ where P is the predicate *squared is equal to 4* and x is the variable.

P is referred to as the *Propositional Function*.

As soon as a value is assigned to the variable x, the statement $P(x)$ becomes a proposition and has a truth value.

Example 1 Let x be an integer and let P be the propositional function *square is equal to 4*, therefore $P(2)$ is *TRUE* and $P(3)$ is *FALSE*.

7.2.3 Predicates with multiple variables

It's important to note that Predicates can depend on **more than one** variable.

Example 1 Let $P(x, y)$ denote $x^2 > y$, therefore $P(-2, 3) \equiv 4 > 3$ is *TRUE* and $P(2, 4) \equiv 4 > 4$ is *FALSE*.

Example 2 Let $Q(x, y, z)$ denote $x + y < z$, therefore $Q(2, 4, 5) \equiv 2 + 4 < 5$ which is *FALSE*, $Q(1, 2, 4) \equiv 1 + 2 < 4$ which is *TRUE*, and $Q(1, 2, z)$ is **NOT** a proposition.

7.2.4 Logical operations

All logic previously defined for propositional logic carries over to predicate logic.

Example 1 If $P(x)$ denotes $x^2 < 16$, then $P(1) \vee P(-5) \equiv (1 < 16) \vee (25 < 16) \equiv \mathbf{T} \vee \mathbf{F} \equiv \mathbf{T}$.

7.3 4.105 Quantification

Quantification expresses the extent to which a predicate is true over a range of elements.

The two most most important quantifiers are the universal quantifier $\forall$ and the existential quantifier $\exists$.

There is a third quantifier called the uniqueness quantifier $\exists!$.

Example 1 The following statements give examples of **quantified predicates**.

- *All men are mortal.*
- *Some computers are not connected to the network.*

7.3.1 Universal Quantifier $\forall$

The Universal Quantification of a predicate $P(x)$ is the proposition:

- $P(x)$ is true for all values of x in the universe of discourse.

We use the notation $\forall x P(x)$ and read it as *for all x.*

If the universe of discourse is finite $\{n_1, n_2, \ldots, n_k\}$ then the universal quantifier is the **conjunction** of the propositions over all elements: $\forall x P(x) \equiv P(n_1) \wedge P(n_2) \wedge \ldots \wedge P(n_k)$.

Example 1 Let P, Q denote the following propositional functions of x:

- $P(X)$: x must take a discrete mathematics course
- $Q(X)$: x is a Computer Science student

Where the universe of discourse for both $P(x)$ and $Q(x)$ is all university students. Let's express the following statements symbolically:

- Every CS student must take a course on discrete mathematics.

  $\forall x Q(x) \rightarrow P(x)$
- Everybody must take a discrete mathematics course or be a CS student.

  $\forall x (P(x) \vee Q(x))$
- Everybody must take a discrete mathematics course and be a CS student.

  $\forall x (P(x) \wedge Q(x))$

Example 2 Let's formalise the following statement S

- S: For every x and for every y, $x + y > 10$

  Let $P(x, y)$ be the statement $x + y > 10$, where the universe of discourse is the set of all integers.

  $\forall x \forall y P(x, y) \equiv \forall x, y P(x, y)$

7.3.2 Existential Quantifier $\exists$

The existential quantification of a predicate $P(x)$ is the proposition *There exists a value of x in the universe of discourse, such that $P(x)$ is true.*

If the universe of discourse is finite $\{n_1, n_2, \ldots, n_k\}$, then the existential quantifier is the **disjunction** of the proposition over all elements: $\exists x P(x) \equiv P(n_1) \vee P(n_2) \vee \ldots \vee P(n_k)$.

Example 1 Let $P(x, y)$ denote the statement $x + y = 5$. The expression $\exists x \exists y P(x, y)$ means "There exists a value x and a value y in the universe of discourse such that $x + y = 5$ is true".

Example 2 Let a, b, c denote fixed real numbers and S be the statement /There exists a real solution to $ax^2 + bx - c = 0$. S can be expressed as $\exists x P(x)$.

7.3.3 Uniqueness Quantifier $\exists!$

The uniqueness quantification of a predicate $P(x)$ is the proposition *There exists a unique value x in the universe of discourse such that $P(x)$ is true.*

The uniqueness quantifier is a **special case** of the existential quantifier.

We use the notation $\exists! x P(x)$ and read it as /there exists a unique x".

7.3.4 Example 1

Let $P(x)$ be the statement $x^2 = 4$. The expression $\exists! x P(x)$ means "There exists a unique value of x such that $x^2 = 4$ is true".

7.4 4.107 Nested Quantifiers

When we want to express statements with multiple variables, we employ nested quantifiers.

Nested Quantifier	Meaning
$\forall x \forall y P(x, y)$	$P(x, y)$ is true for every pair (x, y)
$\exists x \exists y P(x, y)$	There is a pair (x, y) for which $P(x,y) is true
$\forall x \exists y P(x, y)$	For all x, there is a y for which P(x,y) is true
$\exists x \forall y P(x, y)$	There is an x for which $P(x, y)$ is true for all y

7.4.1 Binding Variables

A variable is said to be **bound** if it is within the scope of a quantifier. A variable that is **not bound** is called a **free** variable.

Example 1 Let P be a propositional function and S be the statement $\exists x P(x, y)$. In this case, x is **bound** while y is **free**.

7.4.2 Logical operations

All the logical operations discussed previously, can also be applied to quantified statements.

7.4.3 Order of operations

When we have quantifiers of the same type, either all universal or all existential, the other doesn't matter. However, when we're dealing with quantifiers of different types we **must** apply the quantifiers at the correct order.

Example 1 $\forall x \forall y P(x, y) \equiv \forall y \forall x P(x, y)$

However

$\forall x \exists y P(x, y) \neq \exists y \forall x P(x, y)$

7.4.4 Precedence of Quantifiers

The quantifiers $\forall$ and $\exists$ have precedence over **all** other logical operators. This means that $\forall x P(x) \vee Q(x)$ should be read as $(\forall x P(x)) \vee Q(x)$ and $\forall x P(x) \rightarrow Q(x)$ is to be read as $(\forall x P(x)) \rightarrow Q(x)$.

8 Week 8

Learning Objectives

- Identify logical equivalence involving quantifiers and apply De Morgan's laws.
- Apply predicate logic to programming.

8.1 4.201 De Morgan's laws for quantifiers

8.1.1 The intuition of De Morgan's Laws

Negating quantified expressions is a common activity. Starting with an example.

Let S be the statement *All the university's computers are connected to the network*, and P be the statement *There is at least one computer in the university operating on Linux.*

Intuitively we can find contradictions to both S and P. In the case of S, if we find **at least one** computer **not** connected to the network, then we contradict S. In the case of P, if **not a single** computer operates on Linux, we contradict P. Note that the statement *Not a single computer operates on Linux* is equivalent to *All computers are **not** operating on Linux.*

De Morgan's Laws formalise these intuitions.

8.1.2 De Morgan's Laws

Quantified Expression	Negated Expression
$\forall x P(x)$	$\exists x \neg P(x)$
$\exists x P(x)$	$\forall x \neg P(x)$

8.1.3 Example 1

Let S be the statement *Every student of Computer Science has taken a course in Neural Networks.* Therefore S can be expressed symbolically as $\forall x P(x)$ where $U = \{\text{students in CS}\}$ and $P(x) = x$ has taken a course in Neural Networks.

The negation of S is $\neg S$ which translates to *It is **not** the case that every student in Computer Science has taken a course in Neural Networks.* This statement implies that *There is at least one student of Computer Science who has **not** taken a course in Neural Networks.* Which can be expressed symbolically as $\exists x \neg P(x)$.

8.1.4 Negating nested quantifiers

When we have expressions with nested quantifiers, we simply apply De Morgan's successively from left to right.

$$\begin{aligned} \forall x \exists y \forall z P(x, y, z) &\equiv \exists x \neg \exists y \forall z P(x, y, z) \\ &\equiv \exists x \forall y \neg \forall z P(x, y, z) \\ &\equiv \exists x \forall y \exists z \neg P(x, y, z) \end{aligned}$$

8.2 4.203 Rules of inference

8.2.1 Valid argument

In propositional logic, an *argument* is defined as a sequence of propositions. The final proposition is called the *conclusion* and each of the other propositions are called the *premises* (or *hypotheses*).

And *argument* is **valid** if the truth of **all** its premises implies the truth of the *conclusion*.

Example 1 Given the following *argument*:

- If you have access to the internet, you can order a book in Machine Learning.
- You have access to the internet

- $\therefore$ Therefore, you can order a book on Machine Learning.

This argument is **valid** because whenever all its premises are true, the conclusion must also be true.

Example 2 Given the following *argument*:

- If you have access to the internet, you can order a book in Machine Learning.
- You can order a book on Machine Learning.

- $\therefore$ Therefore, you have access to the internet.

This argument is **not valid** because we can imagine situations where both premises are true, but the conclusion is false.

8.2.2 Rules of inference

These can be seen as **building blocks** for constructing complex valid arguments incrementally.

Given an argument, we could use a truth table to decide whether an argument is true or false, however we will need 2^n rows in the truth table for n variables. This process can be really labor-intensive.

Rules of inference, however, provide a much simpler way of proving the validity of an argument.

Moreover, every rule of inference can be proved using a Tautology[1].

8.2.3 Modus Ponens

Here's the tautology used as the basis for *Modus Ponens*.

$$(p \wedge (p \rightarrow q)) \rightarrow q$$

And the rules of inference

$$\begin{array}{l} p \rightarrow q \\ p \\ \hline \therefore q \end{array}$$

And an example

```
p: It is snowing
q: I will study Discrete Mathematics

If it is snowing, I will study Discrete Mathematics.
It is snowing.

Therefore, I will study Discrete Mathematics.
```

8.2.4 Modus Tollens

Tautology shown below is the basis for *Modus Tollens*.

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

And the rule of inference

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

And an example

```
p: It is snowing
q: I will study Discrete Mathematics

I will not study Discrete Mathematics
If it is snowing, I will study Discrete Mathematics.

Therefore, it is not snowing.
```

8.2.5 Conjunction

The following tautology is the basis for *Conjunction*.

$$((p) \wedge (q)) \rightarrow (p \wedge q)$$

This leads to the following rule of inference

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

And an example

```
p: I will study Programming
q: I will study Discrete Mathematics

I will study Programming
```

```
I will study Discrete Mathematics

Therefore, I will study Programming and Discrete Mathematics
```

8.2.6 Simplification

The next tautology is the basis for *Simplification*

$$(p \wedge q) \rightarrow p$$

It leads to the following valid argument form

$$\frac{p \wedge q}{\therefore p}$$

And an example

```
p: I will study Programming
q: I will study Discrete Mathematics

I will study Programming and Discrete Mathematics

Therefore, I will study Programming
```

8.2.7 Addition

The tautology shown below is the basis for *Addition*

$$p \rightarrow (p \vee q)$$

It leads to the following valid argument form

$$\frac{p}{\therefore p \vee q}$$

And an example

```
p: I will study Programming
q: I will study Discrete Mathematics

I will study Programming

Therefore, I will study Programming or Discrete Mathematics
```

8.2.8 Hypothetical Syllogism

The tautology that follows is the basis for *Hypothetical Syllogism*

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

It leads to the following valid argument form

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

And an example

```
p: It is snowing
q: I will study Discrete Mathematics
r: I will get good grades

If it is snowing, then I will study Discrete Mathematics
If I study Discrete Mathematics, then I will get good grades

Therefore, if it is snowing, then I will get good grades
```

8.2.9 Disjunctive Syllogism

The following tautology is the basis for *Disjunctive Syllogism*

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

It leads to the following valid argument form

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

And an example

```
p: I will study Programming
q: I will study Discrete Mathematics

Either I will study Programming or Discrete Mathematics
I will not study Programming

Therefore, I will study Discrete Mathematics
```

8.2.10 Resolution

The next tautology is the basis for *Resolution*

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

It leads to the following valid argument form

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

And an example

```
p: It is raining
q: It is cold
r: It is snowing
```

```
Either it's raining or it's cold
It's not raining or it's snowing

Therefore, It's cold or it's snowing
```

8.2.11 Building valid arguments

1. If it's initially written in English, transform into argument form by assigning variables to each proposition
2. Start with the hypothesis of the argument
3. Build a sequence of steps in which each step follows from the previous by applying **rules of inference** or **laws of logic**
4. The final step is the conclusion

Example 1 Let's build a valid argument from the following premises:

- *It is not cold tonight*
- *We will go to the theatre only if it is cold*
- *If we do not go to the theatre, we will watch a movie at home*
- *If we watch a movie at home, we will need to make popcorn*

First, we define our propositional variables:

- **p**: It is cold tonight
- **q**: We will go to the theatre
- **r**: We will watch a movie at home
- **s**: We will need to make popcorn

Next, we convert English into logical statements using our propositional variables. We have, respectively:

- $\neg p$
- $q \rightarrow p$
- $\neg q \rightarrow r$
- $r \rightarrow s$

Now we can build a valid argument:

	Step	Justification
1	$q \to p$	Hypothesis
2	$\neg p$	Hypothesis
3	$\therefore \neg q$	Modus Tollens 1,2
4	$\neg q \to r$	Hypothesis
5	$\therefore r$	Modus Ponens 3,4
6	$r \to s$	Hypothesis
7	$\therefore s$	Modus Ponens 5,6

Conclusion: We will need to make popcorn

8.2.12 Fallacies

A fallacy is the use of incorrect argument when reasoning. Formal fallacies can be expressed in propositional logic and proved to be incorrect.

Common formal fallacies:

- affirming the consequent
- a conclusion that denies the premises
- contradictory premises
- denying the antecedent
- existential fallacy
- esclusive premises

8.3 4.205 Rules of inference with quantifiers

The rules are:

- Universal Instantiation
- Universal Generalisation
- Existential Instantiation
- Existential Generalisation
- Universal Modus Ponens
- Universal Modus Tollens

These rules will either **reintroduce** or **remove** quantifiers within a statement.

8.3.1 Universal Instantiation (UI)

The rule of inference:

$$\frac{\forall x P(x)}{\therefore P(c)}$$

Example:

All Computer Science students study Discrete Mathematics.

$\therefore$ Therefore John, who is a Computer Science student, studies Discrete Mathematics.

8.3.2 Universal Generalisation (UG)

The following rule of inference is used to conclude that a proposition is valid for all members of the Universe of Discourse by showing that it is valid for an arbitrary element in the Universe of Discourse.

$$\frac{P(c)}{\therefore \forall x P(x)}$$

Example:

Let $DS = \{\text{all data science students}\}$ and let c be an arbitrary element in DS.

c studies machine learning.

$\therefore$ Therefore, $\forall x \in DS$, x studies machine learning.

8.3.3 Existential Instatiation (EI)

This is used to conclude that there is an element c for which $P(c)$ is true.

$$\frac{\exists x P(x)}{\therefore P(c)}$$

Example:

Let $DS = \{\text{all data science students}\}$. There exists a student x of data science who uses Python Pandas Library.

$\therefore$ Therefore, there is a student, c, who is using Python Pandas Library.

8.3.4 Existential Generalization (EG)

The existential generalization is used to conclude that $\exists x P(x)$ when $P(c)$ is true for some c.

$$\frac{P(c)}{\therefore \exists x P(x)}$$

Example:

Let $DS = \{\text{all data science students}\}$. John, a student of data science, got an A in the machine learning course.

$\therefore$ Therefore, there exists someone in DS who got an A in machine learning.

8.3.5 Universal Modus Ponens

The Universal Modus Ponens can be thought of as a combination of Universal Instantiation and Modus Ponens.

$$\frac{\begin{array}{l}\forall x P(x) \to Q(x) \\ P(a)\end{array}}{\therefore Q(a)}$$

Example:

Let $DS = \{\text{all data science students}\}$. Every computer science student studying data science, will study machine learning.

John is studying data science.

$\therefore$ John will study machine learning.

8.3.6 Universal Modus Tollens

Similarly, this can be seen as a combination of Universal Instantiation and Modus Tollens.

$$\frac{\begin{array}{l}\forall x P(x) \to Q(x) \\ \neg Q(a)\end{array}}{\therefore \neg P(a)}$$

Example:

Let $DS = \{\text{all data science students}\}$. Every computer science student studying data science, will study machine learning.

John is not studying machine learning.

$\therefore$ John is not studying data science.

8.3.7 Expressing complex statements

Given a statement in natural language, we can formalise it by following these steps:

1. Determine the universe of discourse of the variables
2. Reformulate the statement by making **for all** and **there exists** explicit
3. Reformulate the statement by introducing **variables** and defining **predicates**.

4. Reformulate the statement by introducing **quantifiers** and **logical operations**.

Example 1 Express the following statement S: *There exists a real number between any two not equal real numbers.*

1. The universe of discourse is the set of all real numbers $\mathbb{R}$.
2. For all real numbers, there exists a real number between any other two real numbers.
3. For all real numbers x and y, there exists a real number z between x and y.
4. $\forall x, y \in \mathbb{R}\, \exists z \in \mathbb{R}\, x < z < y$

Example 2 Express the following statement S: *Every student has taken a course in machine learning.*

1. The universe of discourse is the set of all students CS. Let $M(x)$ be the proposition *has taken a course in machine learning.*
2. For all students, every student has taken a course in machine learning.
3. For all students x, x has taken a course in machine learning.
4. $\forall x\, M(x)$.

If the Universe of discourse changes, the expression changes:

1. The universe of discourse is the set of all people. Let $S(x)$ be the proposition *is a student* and $M(x)$ be the proposition *has taken a course in machine learning.*
2. S can be expressed as $\forall x(S(x) \to M(x))$.

9 Week 9

Learning Objectives

- Write Boolean expressions.
- Use the laws of Boolean algebra to simplify Boolean expressions.
- Represent a Boolean function.
- Simplify logic circuits/expressions.
- Convert truth tables into Boolean expressions and vice versa.

9.1 5.101 Introduction to Boolean algebra

9.1.1 History of Boolean Algebra

Between 384-322C Aristotle established the foundation of Logic in his *Organon*. Later in 1854, George Boole, an English Mathematician and Logician, published An Investigation of the Laws of Thought on Which are Founded the Mathematical Theories of Logic and Probabilities, which developed a system of *Logical Algebra* by which reasoning can be expressed mathematically.

In 1904, Edward V. Huntington, an American Mathematician, published Sets Of Independent Postulates For The Algebra Of Logic in which he defined 6 rules (or postulates) of Boolean Algebra.

In 1938, Claude Shannon published his Master's Thesis entitled A Symbolic Analysis of Relay and Switching Circuits. In this thesis, Shannon proved that Boolean Algebra could be used to simplify the arrangement of relays; essentially demonstrating that Boolean Algebra could be used to simplify logic.

9.1.2 Applications of Boolean Algebra

Boolean Algebra is the basic building block of computer circuit analysis.

System requirements can be converted into Boolean Algebra and implemented as a circuit. For example, if we have the requirement:

- *When the system is activated, a fire sprinkler should spray water if high heat is detected* could be implemented as follows:

  $$w = hANDa$$

  where h represents *high heat detected*, a represents *system activated* and w represents *spraying water*.

9.1.3 Two-valued Boolean Algebra

The most well-known and widely used for of Boolean Algebra. Variables are binary, therefore they can only accept values of the set $\{0, 1\}$. The operators $+$ and $\cdot$ correspond to OR and AND respectively.

This form of Boolean Algebra can be used to specify and design digital circuits.

9.1.4 Operations of Boolean Algebra

There are three fundamental operations in Boolean Algebra.

- **AND**

  Also referred to as *logical product*, *intersection* or *conjunction*. Can be represented as $x \cdot y$, $x \cap y$ or $x \wedge y$.

  Truth Table is as follows:

x	y	$x \cdot y$
0	0	0
0	1	0
1	0	0
1	1	1

- **OR**

  Also referred to as *logical sum*, *union* or *disjunction*. Can be represented as $x + y$, $x \cup y$ or $x \vee y$.

  Truth Table is as follows:

x	y	x_y
0	0	0
0	1	1
1	0	1
1	1	1

- **NOT**

  Also referred to as *logical complement* or *negation*. Can be represented as x', $\bar{x}$ or $\neg x$.

  Truth Table is as follows:

x	$\bar{x}$
0	1
1	0

When parentheses are not used, these three operators have the following order of precedence: **NOT** $>$ **AND** $>$ **OR**. Note that this still respects the same order of precedence defined in Precedence of logical operators.

9.2 5.103 Postulates of Boolean algebra

9.2.1 Huntington's postulates

There 6 axioms defined by Huntington's postulates. These 6 axioms must be satisfied by any Boolean Algebra.

The axioms are:

- **Closure**

  any result from logical operation must belong to the set $\{0, 1\}$.

- **Identity**

  The logical sum identity $x+0 = x$ and the logical product identity $x \cdot 1 = x$.

- **Commutativity**

  Both logical sum and logical product are commutative, therefore $x + y = y + x$ and $x \cdot y = y \cdot x$.

- **Distributivity**

  Both logical sum and logical product and distributive between each other, therefore $x \cdot (y + z) = x \cdot y + x \cdot z$ and $x + (y \cdot z) = (x + y) \cdot (x + z)$.

- **Complement**

  Complements exist for all elements: $x + \bar{x} = 1$ and $x \cdot \bar{x} = 0$.

- **Distinct Elements**

  This states that any Boolean Algebra has to have two distinct values, therefore $0 \neq 1$.

9.2.2 Basic theorems

Using the 6 axioms from previous sections, we can establish other useful theorems for designing and analysing digital circuits.

Logical Sum	Theorem	Logical Product
$x + x = x$	**Idempotent Laws**	$x \cdot x = x$
$x + 1 = 1$	**Tautology and Contradiction**	$x \cdot 0 = 0$
$\bar{\bar{x}} = x$	**Involution**	
$(x + y) + z = x + (y + z)$	**Associative Laws**	$(x \cdot y) \cdot z = x \cdot (y \cdot z)$
$x + (x \cdot y) = x$	**Absorption Laws**	$x \cdot (x + y) = z$
if $y + x = 1$ and $y \cdot x = 0$, then $x = \bar{y}$	**Uniqueness of Complement**	
$\bar{0} = 1$ and $\bar{1} = 0$	**Inversion Law**	

9.2.3 De Morgan's Theorems

Theorem 1 (De Morgan's Theorem 1)**.** *The complement of a logical product or variables is equal to the logical sum of the complements of the variables.* $\overline{x \cdot y} = \bar{x} + \bar{y}$.
Theorem 2 (De Morgan's Theorem 2)**.** *The complement of a logical sum or variables is equal to the logical product of the complements of the variables.* $\overline{x + y} = \bar{x} \cdot \bar{y}$.

9.2.4 Principle of duality

Starting with a Boolean Relation, we can build another equivalent Boolean Relation by:

- Changing every $+$ to $\cdot$
- Changing every $\cdot$ to $+$
- Changing every 0 to 1
- Changing every 1 to 0

For example $A + B \cdot C \equiv A \cdot (B + C)$.

9.2.5 Ways of proving theorems

There are 4 ways to prove the equivalence of Boolean relations

- **Perfect Induction**

  Show that both relations have identical truth tables. This can be tedious as the number of variables grow.

- **Axiomatic Proof**

  Apply Huntington's postulates or theorems to the expressions until identical expressions are found.

- **Duality Principle**

  Every theorem remain valid after application of the Duality Principle.

- **Contradiction**

  Assuming the hypothesis is false and then proving that the conclusion is also false.

Example: Absorption Rule This rule can be proved easily with a truth table:

x	y	$x + (x \cdot y)$
0	0	0
0	1	0
1	0	1
1	1	1

It can also be proved directly:

$$\begin{aligned} x + (x \cdot y) &\equiv (x \cdot 1) + (x \cdot y) \\ &\equiv x \cdot (1 + y) \\ &\equiv x \cdot 1 \\ &\equiv x \end{aligned}$$

To prove the second part of the absorption law, we can use the duality principle, therefore $x + (x \cdot y) = x = x \cdot (x + y)$.

9.3 5.105 Boolean functions

9.3.1 Definition

Definition 22 (Boolean Function)**.** *A Boolean Function defines a mapping from one or multiple Boolean input values to a Boolean output value.*

For n input values, there are 2^n possible combinations.

Given a function f with 3 input values, f can be completely defined with a table of 8 rows:

x	y	z	$f(x,y,z)$
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

9.3.2 Algebraic forms

There is a single way of representing a Boolean function in a truth table, however algebraically a function can be represented in a variety of forms. For example $f(x) = x + \bar{x} \cdot y = x + y$.

9.3.3 Standardised forms of a function

Boolean functions are standardised to be represented either a *sum-of-products* or *product-of-sums*.

The *sum-of-products* is a function of the form $f(x,y,z) = xy + xz + yz$, while a *product-of-sums* is a function of the form $f(x,y,z) = (x+y) \cdot (x+z) \cdot (y+z)$.

The *sum-of-products* form is, usually, easier to use and simplify.

9.3.4 Build a *sum-of-products* form

Building *sum-of-products* from a truth table is simple.

1. Look at values of the variables that make the function evaluate to 1.
2. If an input is 1, it appears *uncomplemented* in the expression
3. If an input is 0, it appears *complemented* in the expression
4. The function f is, then, represented as a *sum-of-products* of all the terms for which the function is 1.

Example Given the truth table below, represent the function as a *sum-of-products*.

x	y	$f(x,y)$
0	0	0
0	1	1
1	0	1
1	1	1

Therefore, $f(x,y) = \bar{x}y + x\bar{y} + xy$.

9.3.5 Useful functions

The *exclusive-or* function $x \oplus y$ is true when either x or y is true, but not both. This can be expressed as $f(x, y) = \bar{x}y + x\bar{y}$.

The *implies* function $x \to y$ is false when x is true and y is false, and true otherwise. This can be expressed $f(x, y) = \bar{x} + y$.

10 Week 10

Learning Objectives

- Write Boolean expressions.
- Use the laws of Boolean algebra to simplify Boolean expressions.
- Represent a Boolean function.
- Simplify logic circuits/expressions.
- Convert truth tables into Boolean expressions and vice versa.

10.1 5.201 Logic gates

10.1.1 Definition of a gate

A Logic Gate is the basic element of circuits implementing a Boolean operation. The most elementary of such gates are the **OR** gate, the **AND** gate and the **NOT** (or inverter) gate.

All Boolean functions can be written in terms of these three logic gates.

10.1.2 AND Gate

The AND Gate produces a *HIGH* output (value 1) when all its inputs are *HIGH*; otherwise, the output *LOW* (value 0).

The AND Gate is represented as show in figure 1, below we show its truth table:

x	y	$x \cdot y$
0	0	0
0	1	0
1	0	0
1	1	1

Note that $f = x \cdot y$ can also be written as $f = xy$.

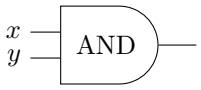


Figure 1: A 2-input AND Gate

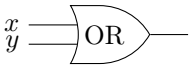


Figure 2: A 2-input OR Gate

10.1.3 OR Gate

The OR Gate produces a *HIGH* output (value 1) when at least one of its inputs are *HIGH*; otherwise, the output *LOW* (value 0).

The OR Gate is represented as show in figure 2, below we show its truth table:

x	y	$x + y$
0	0	0
0	1	1
1	0	1
1	1	1

10.1.4 Inverter Gate

The Inverter Gate produces a *HIGH* output (value 1) when its input is *LOW*; and produces a *LOW* when its input is *HIGH*.

The Inverter Gate is represented as show in figure 3, below we show its truth table:

x	$\bar{x}$
0	1
1	0

10.1.5 XOR Gate

The XOR Gate produces a *HIGH* output (value 1) its inputs have different values and a *LOW* otherwise.

The XOR Gate is represented as show in figure 4, below we show its truth table:

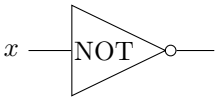


Figure 3: An Inverter Gate

Figure 4: A 2-input XOR Gate

Figure 5: A 2-input NAND Gate

x	y	$x \oplus y$
0	0	0
0	1	1
1	0	1
1	1	0

10.1.6 NAND Gate

Equivalent to "not AND", so this is the inversion of the AND Gate. The NAND Gate is represented as show in figure 5.

10.1.7 NOR Gate

Similarly, this is the inversion of the OR Gate. The NOR Gate is represented as show in figure 6.

10.1.8 XNOR Gate

The inversion of the XOR Gate. The XOR Gate is represented as show in figure 7.

10.1.9 Multiple input gates

AND, OR, XOR and XNOR are all **commutative** and **associative** operations. All of them can be extended to more than 2 inputs as shown in figure 8.

NAND and NOR are both **commutative** but **not associative**. Extending number of inputs is less obvious. We must use parentheses correctly when cascading NAND and NOR operations.

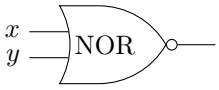


Figure 6: A 2-input NOR Gate

Figure 7: A 2-input XNOR Gate

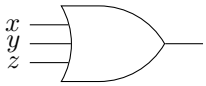


Figure 8: A 3-input OR Gate

10.1.10 Representing De Morgan's Laws

- **Theorem 1**

  $\overline{x \cdot y} \equiv \bar{x} + \bar{y}$.

  This says that figure 9 is equivalent to 10.

- * Theorem 2*

  This says that figure 11 is equivalent to 12.

10.2 5.203 Combinational circuits

10.2.1 Definition of circuit

A *Combinational Circuit* is a combination of different logic gates designed to model a Boolean function. In other words, a Combinational Circuit **implements** a particular Boolean function.

10.2.2 Building a circuit from a function

Given a particular boolean function, we can implement a logic circuit representing **all** states of the function.

Intuitively, we want to **minimise** the number of logic gates used in order to minimize the **cost** of producing the circuit.

For example, let's build a circuit for the function f defined as $f(x, y, z) = x + \bar{y}z$. See figure 13 for the result.

Figure 9: A 2-input NAND Gate

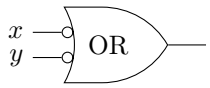


Figure 10: A 2-input OR Gate with negated inputs

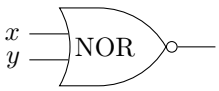


Figure 11: A 2-input NOR Gate

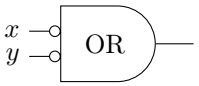


Figure 12: A 2-input AND Gate with negated inputs

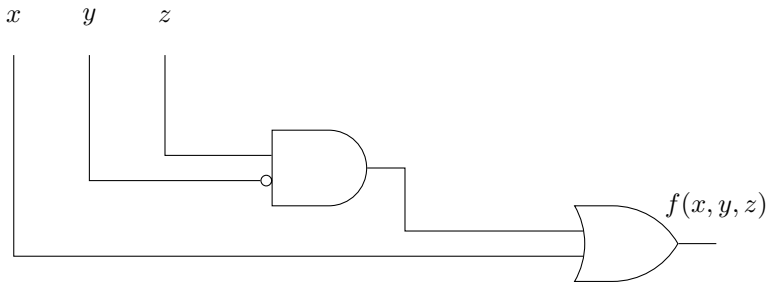


Figure 13: $f(x, y, z) = x + \bar{y}z$

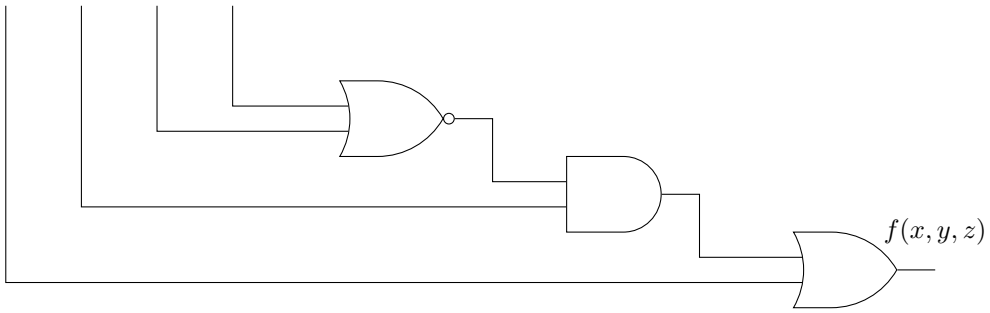


Figure 14: Logic Circuit to extract Boolean expression

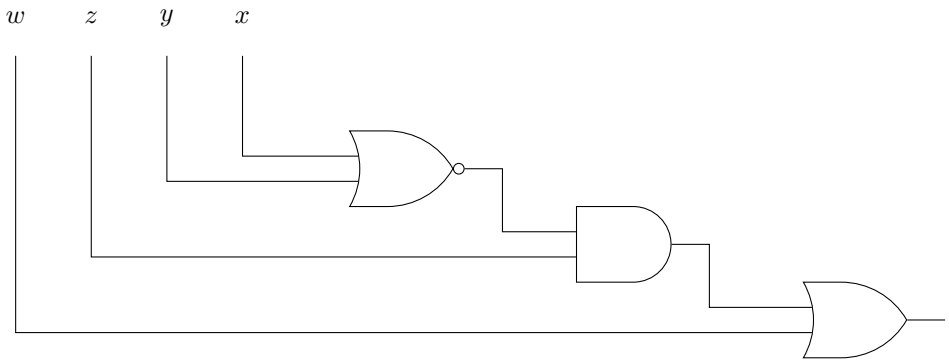


Figure 15: Logic Circuit: Label Inputs

10.2.3 Writing Boolean expressions from a circuit

Given a logic network, we can write its Boolean expression as follows:

1. **Label** all gate outputs that are a function of input variables
2. **Express** the Boolean functions for each gate in the first level
3. **Repeat** until all outputs of the circuit are written as Boolean expressions

Example Consider the circuit in 14. Figures 15, 16, and 17 show the steps necessary to extract a Boolean expression from a circuit layout.

10.2.4 Building a circuit to model a problem

Combinational circuits are useful for designing systems that solve a specific problem. When we want to build a combinational circuit to solve a problem we

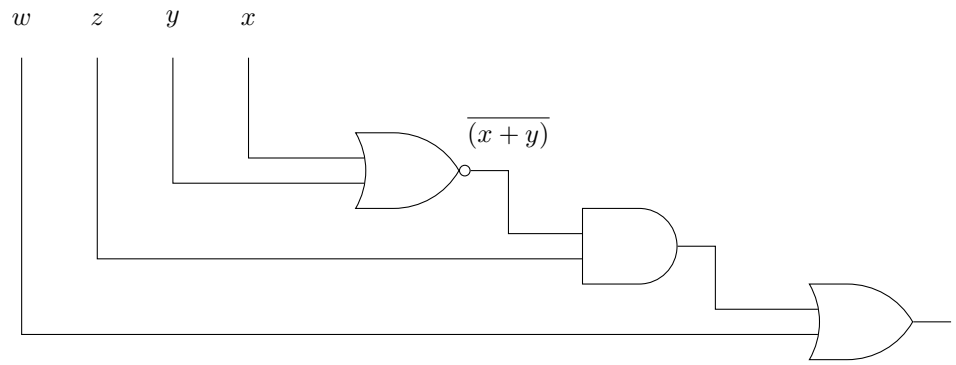


Figure 16: Logic Circuit: Express Boolean Expression of first level

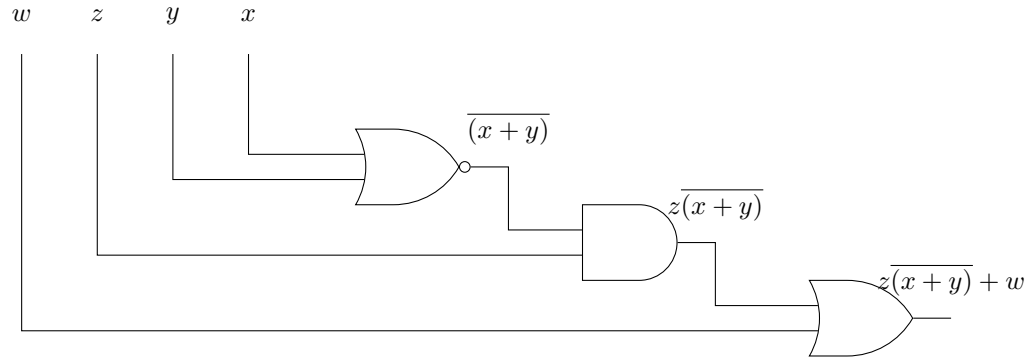


Figure 17: Logic Circuit: Repeat for other levels

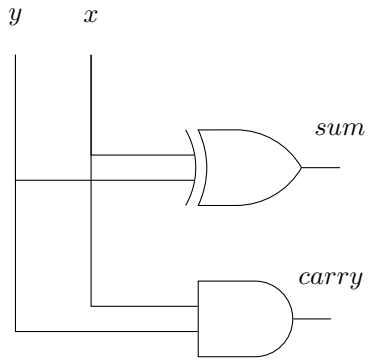


Figure 18: 1-bit half-adder

follow these steps:

1. **Label** inputs and outputs using variables
2. **Model** the problem as Boolean expression
3. **Replace** each operation by equivalent logic gate

10.2.5 Building an adder circuit

We're going to build a Half-adder circuit for two 1-bit inputs and a carry out signal.

The truth table is as follows:

x	y	sum	$carry$
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

If we look closely, the colum *sum* contains the the truth table for an exclusive or gate while the colum *carry* contains the truth table for an and gate. Therefore, our circuit is shown in figure 18.

Note that a half-adder is not useful for multi-bit additions since it lacks a carry input signal.

10.2.6 Building a full adder

To overcome the limitations of a half-adder, we can transform it into a full adder by adding a carry input and including gates for processing the carry input.

The new truth table is as follows:

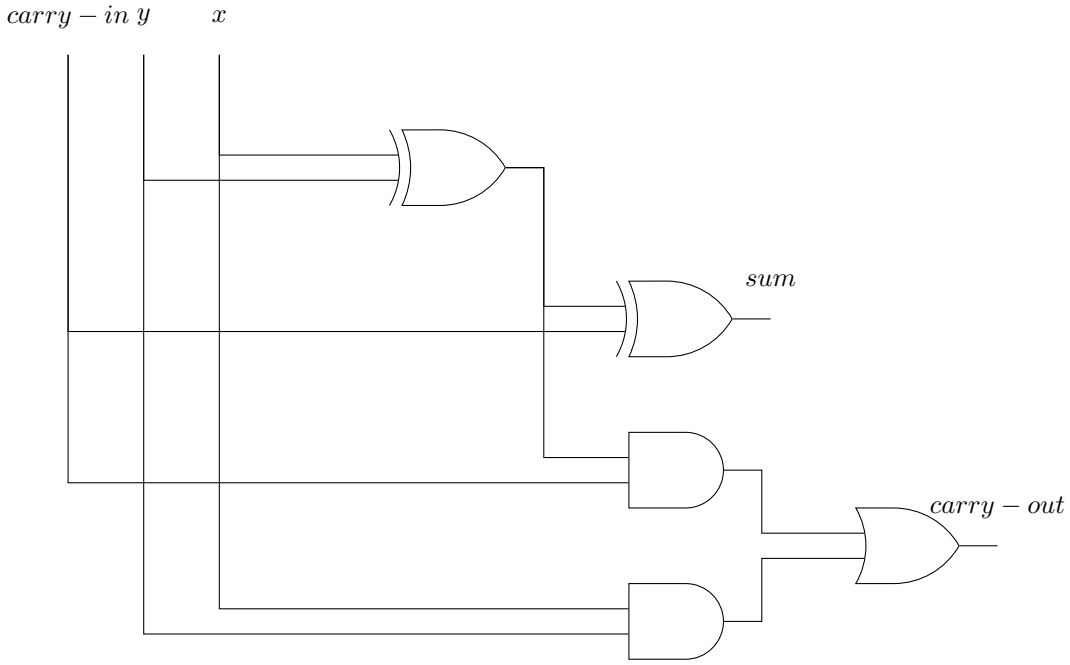


Figure 19: 1-bit full-adder

x	y	$carry - in$	sum	$carry - out$
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

We can see here that $sum = x \oplus y \oplus carry - in$ and $carry - out = xy + carry - in \cdot (x \oplus y)$.

The corresponding circuit is shown in figure 19.

10.3 5.205 Simplification of circuits

10.3.1 Benefits of Simplification

Every function can be written as a sum-of-products, however this may not be the optimal solution in terms of number of gates and the depth of the circuit.

This is why circuits need to be simplified.

Simplification – also referred to as minimisation or optimisation – is benefitial in circuit design for the following reasons:

- Reduces the cost of circuits by reducing the number of gates used
- May reduce the computation time
- Allows more logic to fit into the same area

10.3.2 Algebraic simplification

A technique based on the application of Boolean algebra theorems to simplify the behavior of Boolean functions.

To produce a sum-of-product expression, we usually need to rely on at least one of the following theorems:

- De Morgan's laws and involution
- Distributive laws
- Commutative laws
- Idempotent laws
- Complement laws
- Absorption laws

Example Simplify the following Boolean expression:

$$E = \overline{(\overline{(xy)}z)}\overline{((\bar{x}+z)(\bar{y}+\bar{z}))}$$

$$\begin{aligned}
&\equiv (\overline{\overline{(xy)}} + \bar{z})\overline{((\bar{x}+z)(\bar{y}+\bar{z}))} \\
&\equiv (xy + \bar{z})(\overline{(\bar{x}+z)} + \overline{(\bar{y}+\bar{z})}) \\
&\equiv (xy + \bar{z})(\bar{\bar{x}}\bar{z} + \bar{\bar{y}}\bar{\bar{z}}) \\
&\equiv (xy + \bar{z})(x\bar{z} + yz) \\
&\equiv xyx\bar{z} + xyyz + \bar{z}x\bar{z} + \bar{z}yz \\
&\equiv xy\bar{z} + xyz + x\bar{z} \\
&\equiv xy + x\bar{z}
\end{aligned}$$

10.3.3 Karnaugh maps

A Karnaugh Map (or K-map) is a graphical representation of Boolean functions and is different from a truth table. Adjacent cells in a K-map only change one variable.

	$\bar{y}$	y
$\bar{x}$	0	0
x	1	1

Example Consider the Boolean function represented by the truth table below:

x	y	z	f
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

Since we have three variables, we need a K-map of three inputs:

	$\bar{y}\bar{z}$	$\bar{y}z$	yz	$y\bar{z}$
$\bar{x}$	0	0	1	0
x	1	1	1	1

In this case, the expression is $x + yz$.

11 Week 11

Learning Objectives

- State the principle of mathematical induction.
- Discuss the ideas of the base step and inductive step in a proof by mathematical induction.
- Apply the ideas of mathematical induction to recursion and recursively defined structures.

11.1 6.101 Introduction to proofs

We start with some definitions and terminology and how to formalize a theorem. Then we continue with three types of proofs: direct proof, proof by contraposition, and proof by contradiction.

A proof is, simply put, a valid argument used to prove the truth of a mathematical statement. In order to build a proof, we must rely on all the knowledge introduced previously:

- Variables
- Predicates
- Quantifiers
- Laws of Logic
- Rules of inference

11.1.1 Terminology

Theorem A formal statement that can be shown to be true

Axiom A statement assumed to be true to serve as a premise to further arguments

Lemma A proven statement used as a step to a larger result rather than as a statement of interest by itself

Corollary A theorem that can be established by a short proof from a theorem

11.1.2 Formalising a theorem

The statement **S** *There exists a real number between any two non-equal real numbers* can be formalized as:
Theorem 3. $\forall x, y \in \mathbb{R} x < y \rightarrow \exists z \in \mathbb{R} \mid x < z < y$

11.1.3 Direct proof

A Direct Proof is simply a demonstration that a conditional statement $p \rightarrow q$ is true.

We always with the assumption that p is true, then we employ **axioms**, **definitions**, and **theorems**, together with **rules of inference**, to show that q must also be true.

For example, let's prove our theorem defined in section Formalising a theorem.

Proof. Let x and y be arbitrary elements in $\mathbb{R}$. Let's assume $x < y$.

Let $z = \frac{x+y}{2}$. $z \in \mathbb{R}$, satisfying $x < z < y$.

Using the Universal Generalization rule of inference, we can conclude:

$\therefore \forall x, y \in \mathbb{R} x < y \rightarrow \exists z \in \mathbb{R} \mid x < z < y$ □

11.1.4 Proof by contrapositive

A Proof by Contrapositive is a proof technique that relies on the fact that proving $p \rightarrow q$ is equivalent to proving $\neg q \rightarrow \neg p$.

We start the proof by assuming $\neg q$ is true, then use **axioms**, **definitions**, and **theorems**, together with **rules of inference**, to show that $\neg p$ must also be true.

For example, let's prove the theorem *If n^2 is even, then n is even.*

Proof. Let's assume n is odd. Then $\exists k \in \mathbb{Z} \mid n = 2k + 1$. Then $\exists k \in \mathbb{Z} \mid n^2 = (2k + 1)^2 = 2(2k^2 + 2k) + 1$. Then, n^2 is also odd. Therefore, we conclude that if n is odd, then n^2 is also odd. □

11.1.5 Proof by contradiction

A Proof by Contradiction is a form of proof which relies on assuming the premise to be false and showing that it leads to a contradiction.

We start the proof by assuming $\neg p$ to be true\ then use **axioms**, **definitions**, and **theorems**, together with **rules of inference**, to show that $\neg p$ is false. We can conclude, therefore, that assuming $\neg p$ was wrong, so it must be true.

For example, let's prove the theorem *There are infinitely many prime numbers.*

Proof. Let's suppose there are finitely many prime numbers and list them as $p_1, p_2, p_3, \ldots, p_n$.

Let's consider the number $c = p_1 \cdot p_2 \cdot p_3 \cdot \ldots \cdot p_n + 1$, the product of all primes + 1.

As c is a natural number, it has at least one prime divisor. Then, $\exists k \in \{1, \ldots, n\} \mid p_k/c$. Then, $\exists k \in \{1, \ldots, n\} \exists d \in \mathbb{N} \mid d \cdot p_k = c = p_1 \cdot p_2 \cdot p_3 \cdot \ldots \cdot p_n + 1$.

Then, $\exists k \in \{1, \ldots, n\} \exists d \in \mathbb{N} \mid d = p_1 \cdot p_2 \cdot p_3 \cdot \ldots \cdot p_{k-1} + p_{k+1} + \ldots + p_n + \frac{1}{p_k}$. $\square$

11.2 6.103 The principle of mathematical induction

Mathematical Induction can be used to assert that a propositional function $P(n)$ is true for all positive integers n. In other words, given a propositional function $P(n)$, we can use mathematical induction to show that $P(n)$ holds for all n.

It can formalised using the following rule of inference:

$$\frac{\begin{array}{l} P(1) \text{is true} \\ \forall k(P(k) \rightarrow P(k+1)) \end{array}}{\therefore \forall n P(n)}$$

11.2.1 The intuition behind induction

Let $P(n)$ be the propositional function verifying:

- $P(1)$ is true
- $\forall k(P(k) \rightarrow P(k+1))$

Intuitively, we can say that P is true for 1. Since P is true for 1, then it is true for 2. Since it is true for 2, then it is true for 3, and so on. Since P is true for $n-1$, it's true for n.

What this means is that the **Base Case** $P(1)$ shows that the property holds true initially. The **Inductive Step** $\forall k(P(k) \rightarrow P(k+1))$ shows tha the property holds for all k by showing how each iteration influences the next.

11.2.2 Structure of induction

In order to complete a proof by induction for a propositional function $P(n)$, we need to verify two steps:

Base Case Prove that $P(1)$ is true

Inductive Step Prove that $\forall k \in \mathbb{N} P(k) \rightarrow P(k+1)$[3]

11.2.3 Some uses of induction

Mathematical induction can be used to proce that $P(n)$ is true for all integers greater than a particular threshold, where $P(n)$ is a propositional function. This might cover multiple cases:

- Proving formulas
- Proving inequalities
- Proving divisibility
- Proving properties of subsets and their cardinality

11.3 6.106 Proof by induction

Mathematical Induction is a proof technique which allows us to prove that a property holds for all natural numbers.

11.3.1 Proving formulas

Let's prove $P(n) : 1 + 2 + 3 + \ldots + n = \frac{n(n+1)}{2}$.

Proof. We start with the base case:

$$
\begin{aligned}
1 &= \frac{1(1+1)}{2} \\
1 &= \frac{1(2)}{2} \\
1 &= \frac{2}{2} \\
1 &= 1
\end{aligned}
$$

Then we move on to the inductive step:

[3] $P(k)$ is referred to as the **inductive hypothesis**

$$
\begin{aligned}
1+2+3+\ldots+k &= \frac{k(k+1)}{2} \\
1+2+3+\ldots+k+(k+1) &= \frac{(k+1)(k+2)}{2} \\
\frac{k(k+1)}{2}+(k+1) &= \frac{(k+1)(k+2)}{2} \\
\frac{k(k+1)+2(k+1)}{2} &= \frac{(k+1)(k+2)}{2} \\
\frac{(k+1)(k+2)}{2} &= \frac{(k+1)(k+2)}{2}
\end{aligned}
$$

We have shown that the property holds for the base case and for the inductive step, therefore the proof is complete. □

11.3.2 Proving inequalities

Let's prove $P(n) : 3^n < n!$ if $n \geq 7$.

Proof. We start with the base case:

$$
\begin{aligned}
3^7 &< 7! \\
2187 &< 5040
\end{aligned}
$$

Then we move on to the inductive step.

$$
\begin{aligned}
3^k &< k! \\
3^{k+1} &< (k+1)! \\
3 \cdot 3^k &< (k+1) \cdot k!
\end{aligned}
$$

We have shown that the property holds for the base case and for the inductive step, therefore the proof is complete. □

11.3.3 Proving divisility

Let's prove $P(n) : \forall n \in \mathbb{N} 5 | 6^n + 4$.

Proof. We start with the base case:

$$
\begin{aligned}
5 | 6^0 + 4 \\
5 | 1 + 4 \\
5 | 5
\end{aligned}
$$

Then we move on to the inductive step.

$$
5 | 6^k + 4
$$

Let $6^k + 4 = 5p$, then $5p - 4 = 6k$.

$$
\begin{aligned}
&5|6^{k+1} + 4 \\
&5|6 \cdot 6^k + 4 \\
&5|6(5p - 4) + 4 \\
&5|30p - 24 + 4 \\
&5|30p - 20 \\
&5|5(6p - 4)
\end{aligned}
$$

We have shown that the property holds for the base case and for the inductive step, therefore the proof is complete. □

11.4 6.108 Strong induction

Strong Induction is a form of mathematical induction which makes the inductive step easier to prove by using a stronger hypothesis. We assume that the property holds for all k less than k + 1.

In other words, we employ the conditional statemente $P(1), P(2), \ldots, P(k) \rightarrow P(k+1)l$

11.4.1 Strong induction

It can formalised using the following rule of inference:

$$\frac{\begin{array}{l} P(1)\text{is true} \\ \forall k(P(1), P(2), \ldots, P(k) \rightarrow P(k+1)) \end{array}}{\therefore \forall n P(n)}$$

It's sometimes called **Complete Induction**.

11.4.2 Example

Let's prove $P(n) : \forall n \in \mathbb{N} \wedge n \geq 2, n$ is divisible by a prime number.

Proof. We start with the base case, which reduces to 2. 2 is a prime number and divides itself.

Then we move on to the inductive step. Let $k \in \mathbb{N}$, greater than 2. If the inductive hypothesis $P(k)$ is true, let's assume $P(2), \ldots, P(k+1)$ is true. Then, $\forall m \in \mathbb{N}$ and $2 \leq m \leq k+1$, $\exists p$ a prime number dividing m.

Here we have two cases:

$k + 2$ **is a prime number** then $k + 2$ is trivially divisible by itself

$k + 2$ **is a composite number** then $\exists m$ dividing $k+2$. Because $2 \leq m \leq k+1$, then $\exists p$ dividing m, which also divides $k + 2$.

We have shown that the property holds for the base case and for the inductive step, therefore the proof is complete. □

11.4.3 Well-ordering property

It is an axiom about $\mathbb{N}$ that we assume to be true. The axioms about $\mathbb{N}$ are as follows:

1. The number 1 is a positive integer
2. If $n \in \mathbb{N}$, then $n + 1$ is also a positive integer
3. Every positive integer other than 1, is the successor of a positive integer
4. The Well-ordering property: every non-empty subset of the set of positive integers has at least one element

The well-ordering property can be used as a tool in building proofs.

11.4.4 Example

Let's reconsider our previous proof.

Proof. Let S be the set of positive integers greater than 1 without a prime divisor.

Suppose S is non-empty. Let n be its smallest element. n cannot be prime since n divides itself if n is prime; i.e. n would be its own prime divisor.

Therefore, n must be composite, which means it must have a divisor d such that $1 < d < n$ Then d must have a prime divisor.

Let p be the prime divisor of d. We know that p/d and d/n, therefore p/n, which contradicts our statement that S is the set of positive integers greater than 1 without a prime divisor.

Therefore, S must be an empty set, which verifies $P(n)$. □

11.4.5 Equivalence of the three concepts

- mathematical induction → well-ordering property
- well-ordering property → strong induction
- strong induction → mathematical induction

All three concepts are equivalent.

12 Week 12

Learning Objectives

- Describe the concept of recursion and give examples of its application.
- Identify the base case and the general case of a recursively defined problem.

12.1 6.201 Recursive definitions

12.1.1 Definition

When we define a mathematical object in terms of itself, this is called Recursion.

12.1.2 Recursively defined functions

A recursively defined function f with domain $\mathbb{N}$ is defined by:

Basis Step initial value of the function

Recursive Step a rule for finding the value of the function at an integer from its previous values.

For example the Fibonacci function can be defined as:

$$\begin{cases} f(0) & 1 \\ f(1) & 1 \\ f(n) & f(n-1) + f(n-2) \end{cases}$$

12.1.3 Recursively defined sets

A recursively defined set S, is defined by:

Basis Step initial elements of the set

Recursive Step a rule for generating new elements from the ones we already have

For example, the set S is recursively defined by:

Basis Step $4 \in S$

Recursive Step $x \in S \land y \in S \rightarrow x + y \in S$

12.1.4 Recursive algorithms

An algorithm is a finite sequence of precise instructions for performing a computation or for solving a problem.

A recursive algorithm is an algorithm which solves a problem by reducing it to a smaller instance of the same problem with smaller input.

For example, here's a recursive algorithm for computing $n!$:

Algorithm 1 Computing $n!$

```
1: function FACTORIAL(n)
2:     if n = 0 then
3:         return 1
4:     return n× FACTORIAL(n − 1)
```

12.2 6.204 Recurrence relations

12.2.1 Definitions

A Recurrence Relation is an equation that defines a sequence based on a rule that produces the next term as a function of the previous.

An infinite sequence is a function from the set of integers to the set of real numbers.

In many cases, it's useful to formalise a problem as a sequence before solving it.

12.2.2 Linear recurrences

A relation in which each term of the sequence is a linear function of earlier terms.

There are two types of linear recurrences:

Linear Homogenous Recurrences $a_n = c_1 a_{n-1} + c_2 a_{n-2} + \ldots + c_k a_{n-k}$. $c_1, \ldots, c_k \in \mathbb{R}$. k is the degree of the relation.

Linear Non-homogenous Recurrences $a_n = c_1 a_{n-1} + c_2 a_{n-2} + \ldots + c_k a_{n-k} + f(n)$. $c_1, \ldots, c_k \in \mathbb{R}$. k is the degree of the relation.

12.2.3 Arithmetic sequences

A sequence is called arithmetic if the **difference** between consecutive terms is a constant c.

12.2.4 Geometric sequences

A sequence is called geometric if the **ratio** between consecutive terms is a constant r.

12.2.5 Divide and conquer recurrence

A divide and conquer algorithm consists of three steps:

- Dividing the problem into smaller subproblems
- Solving each subproblem recursively
- Combining all solutions to find a final solution to the original problem

12.3 6.206 Solving recurrence relations

12.3.1 Solving linear recurrence

Let:

$$a_n = c_1 a_{n-1} + c_2 a_{n-2} + \ldots + c_k a_{n-k}$$

be a linear homogenous recurrence. If a combination of the geometric sequence:

$$a_n = r^n$$

is a solution to this recurrence, it satisfies:

$$r^n = c_1 r^{n-1} + c_2 r^{n-2} + \ldots + c_k rn - k$$

By dividing both sides by r^{n-k} we get:

$$r^k = c_1 r^{k-1} + c_2 r^{k-2} + \ldots + c_k$$

This equation is called the **characteristic equation**. Solving this equation is the first step towards finding a solution to the linear homogenous recurrence.

If r is a solution of the equation with multiplicity p, then the combination:

$$(\alpha + \beta n + \gamma n^2 + \ldots + \mu n^{p-1}) r^n$$

satisfies the recurrece.

12.3.2 Induction for solving recurrence

Sometimes it's easier to solve a recurrence relation using strong induction.

13 Week 13

Learning Objectives

- Define a graph, edges, vertices, parallel edges, loops, cycles and walk, path and connected graphs.
- Describe the degree sequence of a graph and the relation that links the sum of the degree sequence.
- Describe special graphs: simple graphs, complete graphs and r-regular graphs.

13.1 7.101 Introduction

We start studying Graphs. Graphs are discrete structures consisting of vertices and nodes connecting them.

Graph Theory is a branch of discrete mathematics which studies these structures.

13.1.1 Applications of Graphs

- Modeling computer networks
- Modeling road maps
- Solving shortest-path problems between cities
- Assigning jobs to employees
- Distinguishing chemical compound structures
- Modeling molecules

13.2 7.103 Definition of a graph

13.2.1 Graph

A graph G is an ordered triple $G = \{V, E\}$, where V is the set of vertices and E is the set of edges.

13.2.2 Vertex

The basic element of a graph, drawn as a node or a dot. The set of vertices of G is usually denoted by $V(G)$ or simply V. Figure 20 denotes a graph with 3 vertices.

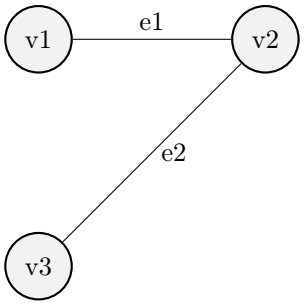


Figure 20: A sample graph with 3 vertices

In the graph shown in figure 20, $V(G) = \{v1, v2, v3\}$.

13.2.3 Edge

A link between two vertices. It's drawn as line connecting exactly two vertices. The set of edges in a graph G is denoted by $E(G)$ or simply E.

In the graph shown in figure 20, $E(G) = \{e1, e2\} = \{\{v1, v2\}, \{v2, v3\}\}$.

13.2.4 Adjacency

- Two vertices are adjacent if they are endpoints on the same edge
- Two edges are adjacent if they share the same vertex
- If vertex v is and endpoint of edge e, then we say that v and e are incident

13.2.5 Loops and parallel edges

Let's consider the following graph:

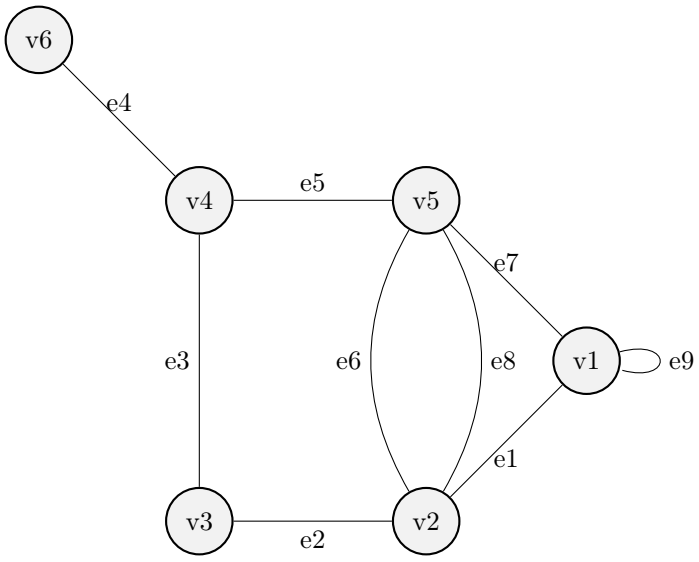


Figure 21: Loops and Parallel Edges

The vertices $v2$ and $v5$ are linked with two edges ($e6$ and $e8$), those edges are referred to as **parallel edges**.

The vertex $v1$ is linked to itself by edge $e9$, that edge is called a **loop**.

13.2.6 Directed Graph - Digraph

A direct graph is a graph in which the edges have a direction, indicated with an arrow on the edge. In the graph shown in figure 22, we can see that $e1$ is a connection from $v1$ to $v2$ but it is **not** a connection from $v2$ to $v1$.

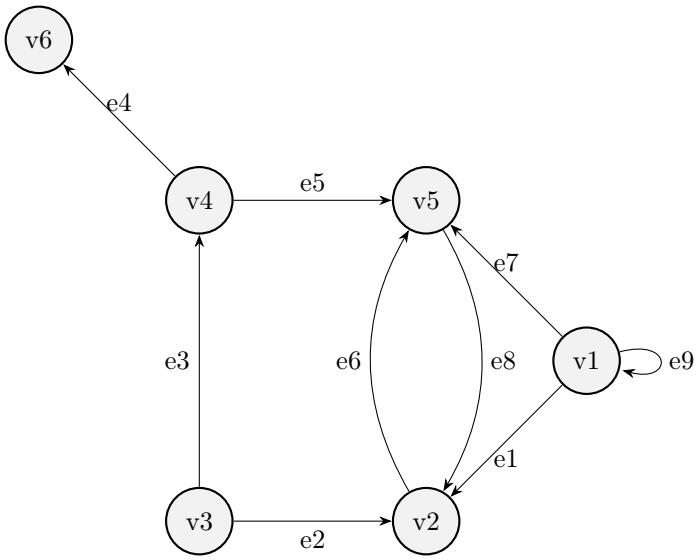


Figure 22: A directed graph

13.3 7.105 Walks and paths in a graph

13.3.1 Definition of a walk

A walk is a sequence of vertices and edges of a graph where vertices and edges can be repeated.

A walk of length k is a succession of k (not necessarily different) edges of the form:

$$uv, vw, wx, \ldots, yz$$

Taking the graph in figure 20 as an example, we can define a walk of length 4 from $v1$ to $v6$:

$$v1v2, v2v3, v3v4, v4v6 = e1, e2, e3, e4 = v1v2v3v4v6$$

13.3.2 Definition of a trail

A trail is a walk in which no edge is ever repeated. Vertices can be repeated, but no edges can be repeated.

13.3.3 Definition of a circuit

A circuit is a closed trail, meaning the starting and ending vertices are the same. Only vertices can be repeated.

13.3.4 Definition of a path

A path is a trail in which neither vertices nor edges are repeated.

13.3.5 Definition of a cycle

A cycle is a closed path in which a vertex is reachable from itself.

13.3.6 Eulerian path

An Eulerian path is a path that uses each edge precisely once. If such a path exists, the graph is called **traversable**.

13.3.7 Hammiltonian path

A Hammiltonian path (or traceable path) is a path that visits each vertex precisely once. If such a path exists, the graph is called a **traceable graph**.

13.3.8 Hammiltonian cycle

A cycle that uses each vertex exactly once (except for the starting vertex, which is visited exactly twice) is called a Hammiltonian cycle. If such a cycle exists, the graph is called a **Hammiltonian graph**.

13.3.9 Connectivity

An **undirected** graph is **connected** if you can get from any node to any other node by following a sequence of edges. This means that any two random nodes in a graph are connected by a path.

The graph depicted in figure 23 is a connected graph while the graph depicted in figure 24 is **not** a connected graph.

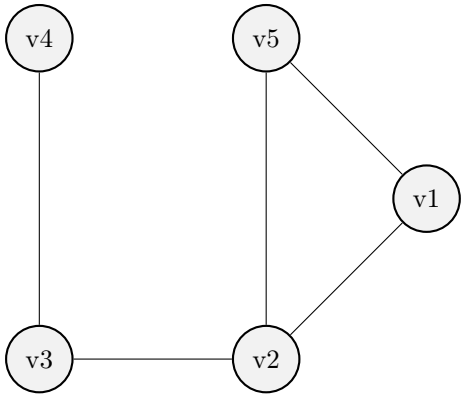


Figure 23: A Connected Graph

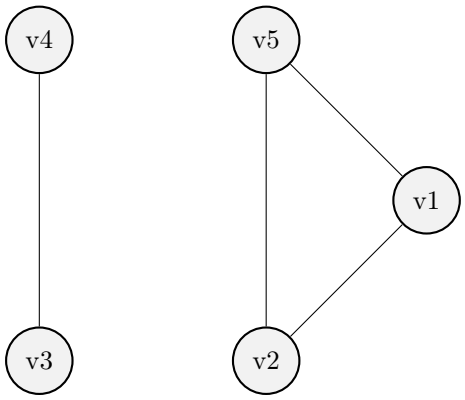


Figure 24: Not A Connected Graph

13.3.10 Strong Connectivity

A directed graph is a strongly connected graph if there is a directed path from any node to any other node. Figure 25 shows a depiction of such a graph. Conversely, the graph depicted by figure 26 is not strongly connected.

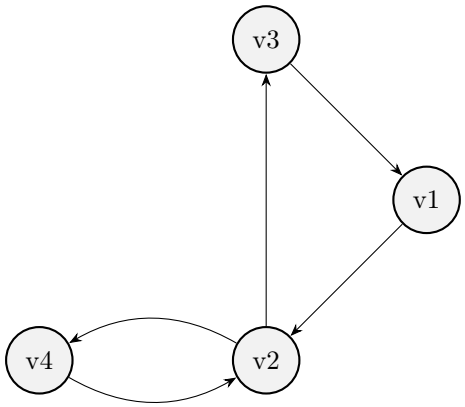


Figure 25: A Strongly Connected Graph

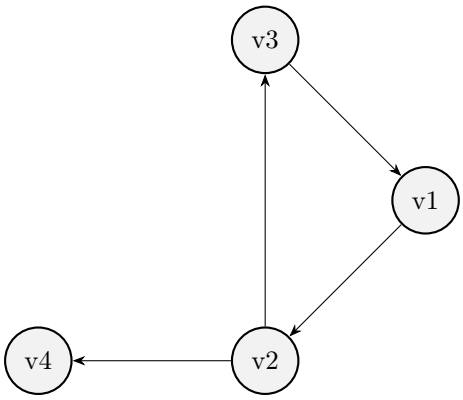


Figure 26: Not A Strongly Connected Graph

13.3.11 Transitive Closure

Given a digraph G, the Transitive Closure of G is the digraph G^* such that:

- G^* has the same vertices as G
- If G has a directed path from u to v ($u \neq v$), G^* has a directed edge from v to u.

Figure 27 shows a depiction of a transitive close of G.

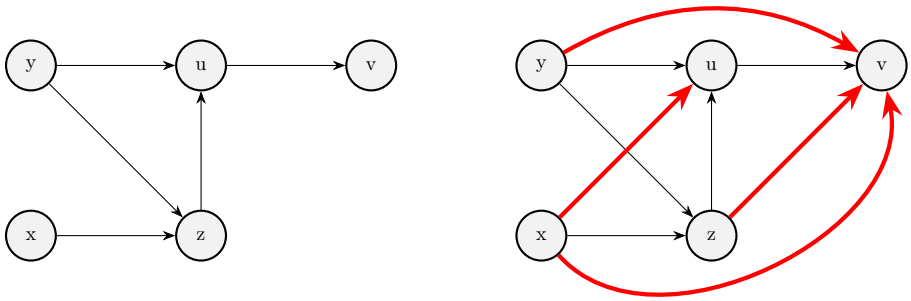


Figure 27: Transitive Closure

13.4 7.107 The degree sequence of a graph

13.4.1 Terminology - Undirected Graphs

Degree ($deg(v)$) the number of edges **incident** on v

- A loop contributes **twice** to the degree

- An isolated vertex has degree 0

In figure 28, we show a graph with the degree for every node written inside the node.

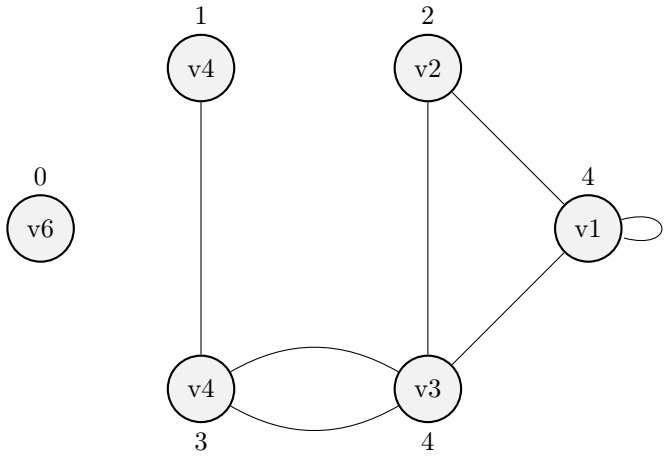


Figure 28: A graph with its vertices' degrees

13.4.2 Terminology - Directed Graphs

In-degree $(in - deg(v))$ Number of edges going **into** v

Out-degree $(out - deg(v))$ Number of edges going **out of** v

Degree $(deg(v))$ $out - deg(v) + in - deg(v)$

In figure 29 we show a graph with all in and out degrees for every vertex.

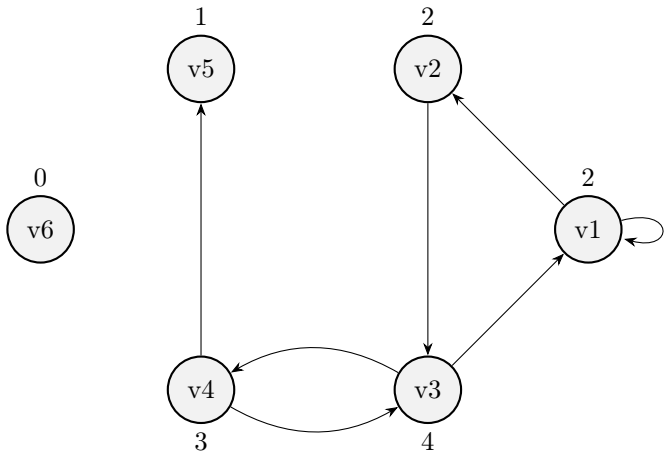


Figure 29: A digraph with its vertices' degrees

13.4.3 Degree sequence of a graph

Given an undirected graph G, a degree sequence is a monotonic non-increasing sequence of the vertex degrees of all the vertices of G.

The degree sequence of the graph from figure 30 is **4,3,2,1**.

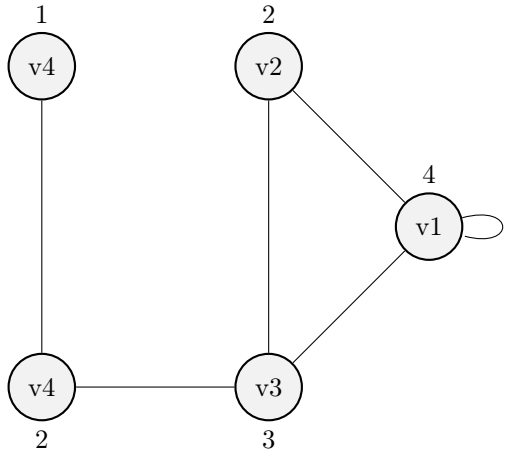


Figure 30: A graph with its degree sequence

13.4.4 Degree sequence property I

- The sum of the degree sequence of a graph is always even

13.4.5 Degree sequence property II

- The sum of the degree sequence of a graph is always twice the number of edges

13.5 7.109 Special graphs: simple, r-regular and complete graphs

13.5.1 Simple Graphs

A graph which contains no loops and no parallel edges, like the one in figure 31.

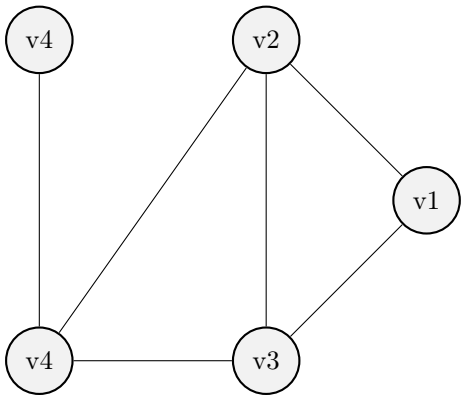


Figure 31: Simple graph

13.5.2 Properties of simple graphs

- Given a simple graph G with n vertices, the degree of each vertex of G is at most $n - 1$

13.5.3 Regular graphs

A graph is said to be regular if all local degrees are the same number.

A graph G where all vertices have the same degree r is called an r-regular graph.

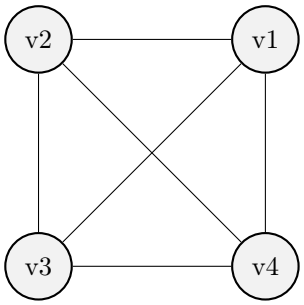


Figure 32: 3-regular graph

13.5.4 Properties of regular graphs

Given an r-regular graph G with n vertices, the following is true:

- Degree sequence of G is $r, r, r, \ldots$ (repeated n times)
- Sum of degree sequence is $r \cdot n$
- Number of edges in G is $\frac{r \cdot n}{2}$

13.5.5 Complete graph

A simple graph where every pair of vertices is adjacent. Complete graphs with n vertices are represented by the symbol K_n.

Figure 33 shows an example of a complete graph with 8 vertices.

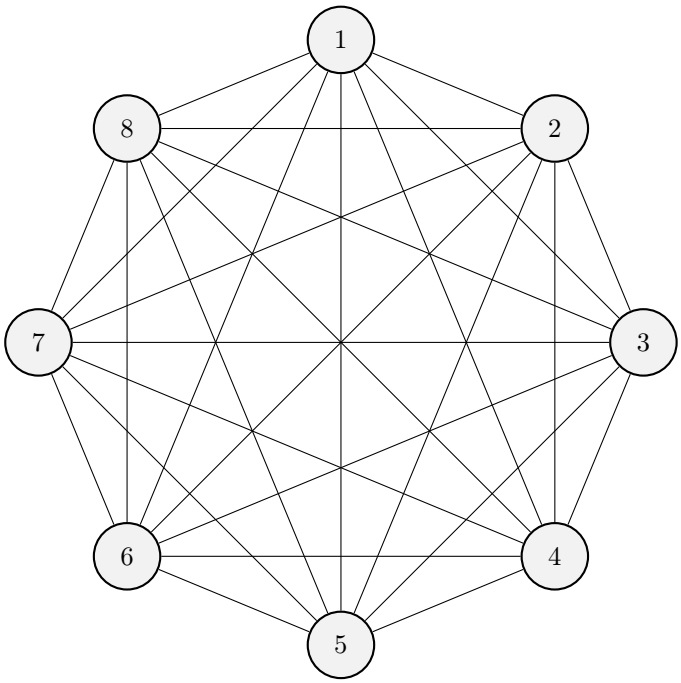


Figure 33: Complete graph

13.5.6 Complete graph properties

A complete graph with n vertices, K_n has the following properties:

- Every vertex has degree $n - 1$
- Sum of degree sequences is $n(n - 1)$
- Number of edges is $\frac{n(n-1)}{2}$

14 Week 14

Learning Objectives

- Define the adjacency matrix of a graph.

14.1 7.201 Isomorphic graphs

14.1.1 Definition of isomorphism

Two graphs G_1 and G_2 are isomorphic if there is a bijection (an invertible function) $f : G_1 \to G_2$ that preserves adjacency and non-adjacency.

This means that if uv is an edge in G_1, then $f(u)f(v)$ is an edge in G_2. In other words, u and v are adjacent in G_1 if and only if $f(u)$ and $f(v)$ are adjacent in G_2.

14.1.2 Properties of isomorphic graphs

- Two graphs with different degree sequences cannot be isomorphic
- Two graphs with the same degree sequence aren't necessarily isomorphic

14.2 7.203 Bipartite graphs

A graph $G(V, E)$ is called bipartite if the set of vertices V can be partitioned in two disjoint sets V_1 and V_2 such that edge e in G has one endpoint in V_1 and one endpoint in $V2$.

Figure 34 depicts a bipartite graph:

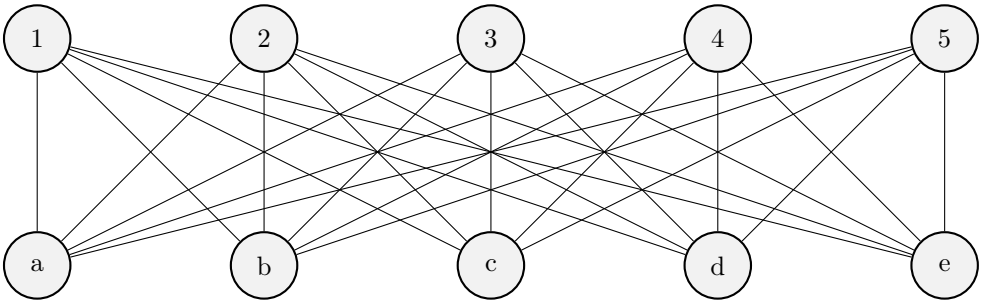


Figure 34: Bipartite Graph

Bipartite graphs are 2-colourable, meaning 2 colours are enough to color the graphs in a way that adjacent nodes do not have the same colour.

There are no odd-length cycles in bipartite graphs.

14.2.1 Matching

A Matching is a set of pairwise non-adjacent edges, none of which are loops. Meaning no two edges share the same endpoint.

A vertex is matched (or saturated) if it is an endpoint of one of the edges in the matching. Otherwise, the vertex is unmatched.

14.2.2 Maximum matching

A matching of maximum size such that if any edge is added, it is no longer a matching.

In a single bipartite graph, there may be many possible maximal matchings.

14.2.3 The Hopcroft-Karp Algorithm 1

The Hopcroft-Karp Algorithm is an algorithm for finding the maximum matching in a bipartite graph.

Augmenting paths a path that starts on a free node, ends on a free node and alternates between unmatched and matched edges within the path

Breadth First Search traversing the graph level by level

Depth First Search traversing the graph all the way to the leaf before starting another path

A simplified pseudocode for the algorithm is as follows:

1. $M \leftarrow \emptyset$
2. While there is an augmenting path P
   (a) Use BFS to build layers that terminate at free vertices
   (b) Start at free vertices in C, use DFS
3. Return M

14.3 7.205 The adjacency matrix of a graph

- Adjacency list
- Adjacency matrix
- Properties of adjacency matrix

14.3.1 Graph representation recap

Graphs can be represented as a set of vertices and a set of edges.

Table 1: Adjancency list

Vertex	Adjacent Vertices
A	B, C
B	A, C, D
C	A, B, D, E
D	B, C, E
E	C, D

14.3.2 Adjacency list

A list of all the vertices of G and their corresponding individual adjacent vertices. Table 1 shows the adjancency list for the graph in figure 35.

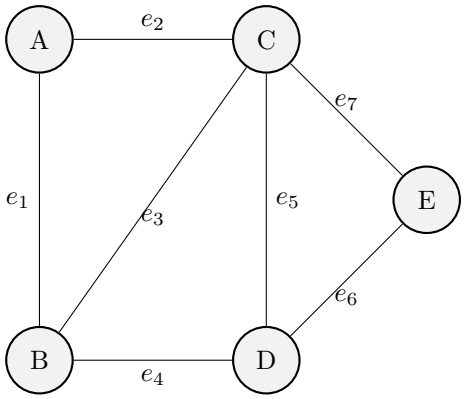


Figure 35: Sample Graph For Adjacency List

Note that we can draw a graph from its adjacency list alone.

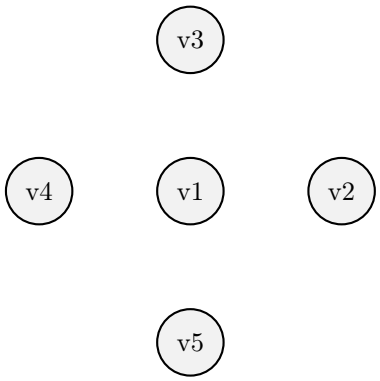
14.3.3 Example

Given the adjacency list in table 2, let's draw the graph:

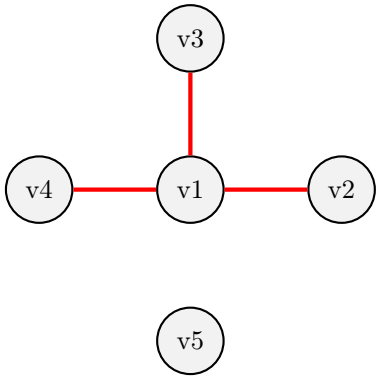
Table 2: Adjancency list

Vertex	Adjacent Vertices
v_1	v_2, v_3, v_4
v_2	v_1
v_3	v_1, v_4, v_5
v_4	v_1, v_3, v_5
v_5	v_3, v_4, v_5

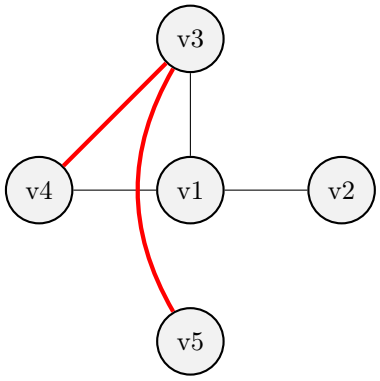
We know, from table 2 that the graph has 5 vertices. So we can start our drawing with only 5 vertices drawn, without any edges. That's shown below:



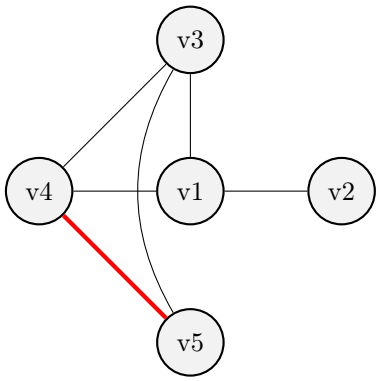
Now we can start adding edges. First covering v_1's adjacent vertices:



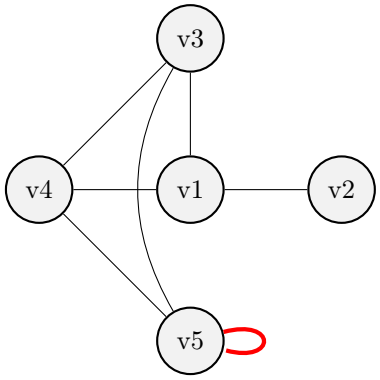
Next we go for v_2, but that's already covered. So we move on to v_3:



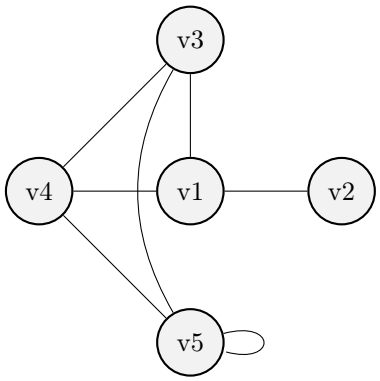
Next, we tackle for v_4:



Lastly, we look at v_5:



Now that all vertices are done, the final graph is:



14.3.4 Adjacency matrix

A graph can be represented by its adjacency matrix. Given the graph is figure 36, we can produce its adjacency matrix as shown below:

$$M(G) = \begin{bmatrix} 2 & 1 & 1 \\ 1 & 2 & 2 \\ 1 & 2 & 0 \end{bmatrix}$$

The format of the matrix is as follows:

$$M(G) = \begin{bmatrix} v_1v_1 & v_1v_2 & v_1v_3 \\ v_2v_1 & v_2v_2 & v_2v_3 \\ v_3v_1 & v_3v_2 & v_3v_3 \end{bmatrix}$$

Note that in the leading diagonal, we have the loops which are counted twice. One interesting observation is that sum of all the edges in the undirected graph, is equal to half the sum of all the elements in its adjacency matrix. The sum of the degree sequence is equal to the sum of the adjacency matrix.

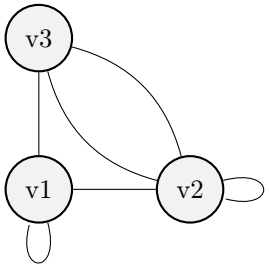


Figure 36: A graph for producing adjacency matrix

The adjacency matrix can also be produced for a directed graph, but we need to in mind that v_1 being adjacent to v_2 does not imply v_2 being adjacent to v_1.

14.4 7.207 Dijkstra's algorithm

14.4.1 Weighted graphs

A weighted graph is a graph in which each edge is assigned a weight. Like shown in figure 37.

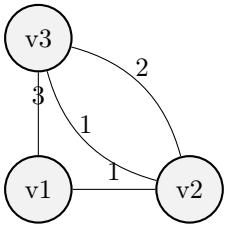


Figure 37: Weighted Graph

It can be used to model the distance between cities, response time in a network or the cost of a transaction, among other things.

14.4.2 Dijstra's algorithm

In 1956, Edsger Dijkstra designed an algorithm for finding the shortest path between any two nodes in a weighted graph. This algorithm is now know as Dijkstra's Algorithm.

15 Week 15

Learning Objectives

- Define a tree.
- Define spanning trees.
- Define non-isomorphic spanning trees.

15.1 8.103 Definition of a tree

15.1.1 Acyclic graph

A graph G is acyclic if and only if it has no cycles.

15.1.2 Definition of a tree

A tree is a connected acyclic undirected graph. Figure 38 depicts a sample tree. A tree cannot have loops or parallel edges.

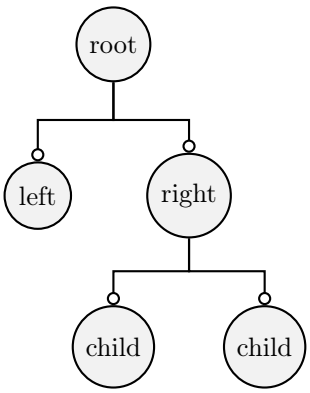


Figure 38: A sample tree

15.1.3 Definition of a forest

A forest is a cycle-free disconnected graph.

15.1.4 Theorem 1

An undirected graph is a Tree if and only if there is a unique simple path between any two of its vertices.

Well, if there are more than one path, we have a cycle.

15.1.5 Theorem 2

A tree with n vertices has $n - 1$ edges.

15.1.6 Rooted trees

A Rooted Tree is a tree in which one vertex has been designated as the **root**. Every edge is directed away from the root.

15.2 8.105 Spanning trees of a graph

A spanning tree of a graph G, is a connected subgraph of G that contains all vertices of G, without any cycles.

Figure 39 shows a complete graph with 4 vertices:

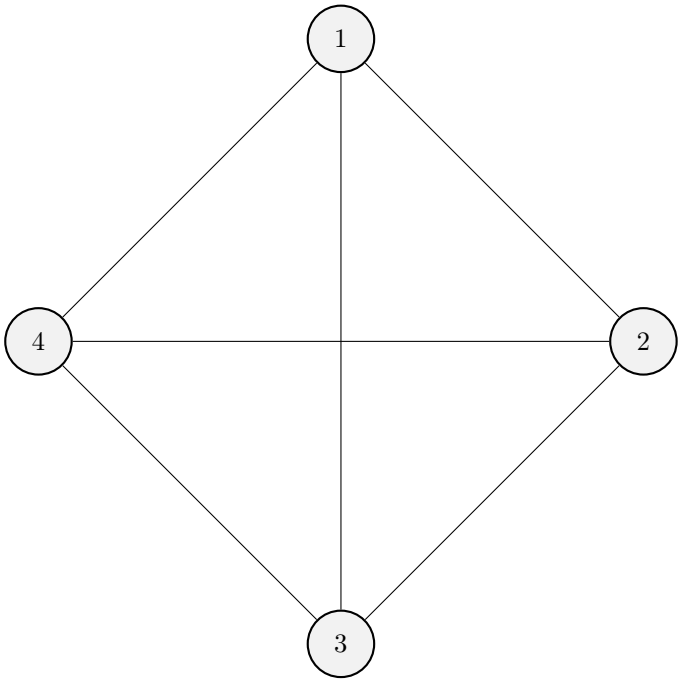


Figure 39: Complete graph with 4 vertices

There are four possible spanning trees in this graph.

15.2.1 Constructing a spanning tree

To get a spanning tree of graph G

1. Keep all vertices of G
2. Break all the cycles but keep the tree connected

Figures 40, 41, 42 show the process in a visual form. Note that figure 42 shows one out of 16 possible spanning trees for the graph shown in figure 40.

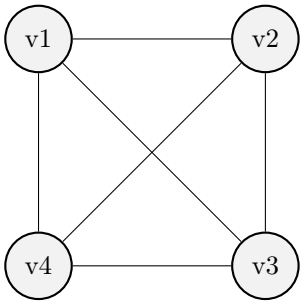


Figure 40: Starting Graph G

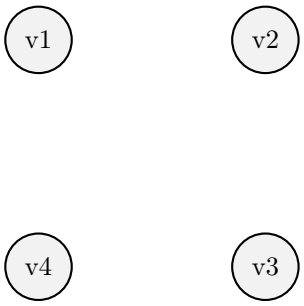


Figure 41: Keep all the vertices

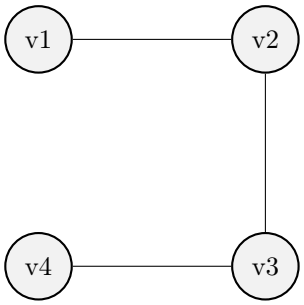


Figure 42: Break all cycles but keep the tree connected

15.2.2 Non iso-morphic spanning trees

Two spanning trees are said to be iso-morphic if there is a bijection preserving adjacency between the two trees.

15.3 8.107 Minimum spanning tree

15.3.1 Spanning tree cost

Suppose we have a connected undirected graph with a weight (or cost) associated with each edge. The cost of a spanning tree would be the sum of the cost of its edges.

The cost of the spanning tree marked in red is $8 + 3 + 2 + 1 = 14$.

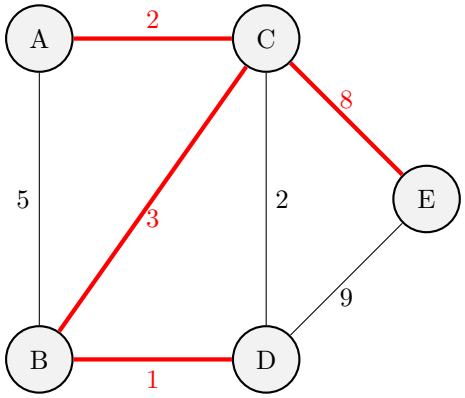


Figure 43: Sample Graph With Weights

15.3.2 Minimum-cost spanning tree

A spanning tree with the lowest cost.

15.3.3 Finding spanning trees

There are two basic algorithms for finding minimum-cost spanning trees.

- Kruskal's Algorithm
- Prim's Algorithm

Both are greedy algorithms.