



# Advanced Information, Computation and Communication

EPFL - IC

**Giovanni Ranieri**

A summary of AICC main concepts

Course given by professors

Karl Aberer

Bixio Rimoldi

Michael Gastpar

# Contents

<b>1</b>	<b>Propositional Logic</b>	<b>11</b>
1.1	Basic Concepts . . . . .	11
1.1.1	Proposition . . . . .	11
1.1.2	Logical Connectives . . . . .	11
1.1.3	Truth Tables . . . . .	11
1.1.4	Terminology . . . . .	12
1.2	Logical Equivalences . . . . .	12
1.2.1	Definition . . . . .	12
1.2.2	Common Mistake . . . . .	12
1.2.3	Tables of Equivalences . . . . .	12
1.3	Morgan's Law . . . . .	13
1.3.1	Definition . . . . .	13
1.3.2	Proof . . . . .	13
1.4	Normal Forms . . . . .	13
1.4.1	Disjunctive Normal Form (DNF) . . . . .	13
1.4.2	Conjunctive Normal Form (CNF) . . . . .	13
1.5	Universal and Existential Quantifiers . . . . .	13
1.5.1	Variables . . . . .	13
1.5.2	Predicates . . . . .	13
1.5.3	Predicates and Propositional Logic . . . . .	14
1.5.4	Propositional Functions . . . . .	14
1.5.5	Universal Quantifier . . . . .	14
1.5.6	Existential Quantifier . . . . .	14
1.6	Validity and Satisfiability . . . . .	14
1.6.1	Statement Valid . . . . .	14
1.6.2	Statement Satisfiable . . . . .	14
1.7	The Morgan's Law for Quantifiers . . . . .	14
1.8	Visualization of Quantifiers . . . . .	15
1.9	Proofs . . . . .	15
1.9.1	Argument . . . . .	15
1.9.2	Inference Rules . . . . .	15
1.9.3	Inference Rules with Quantifiers . . . . .	16
1.9.4	Type of Proofs . . . . .	16
<b>2</b>	<b>Basic Structures: Sets, Functions, Sequences and Sums</b>	<b>17</b>
2.1	Basics of Sets . . . . .	17
2.1.1	Definition . . . . .	17
2.1.2	Set Equality . . . . .	17
2.1.3	Subset . . . . .	17
2.1.4	Proper Subset . . . . .	17
2.1.5	Power Set . . . . .	17
2.1.6	Tuples and Equality of Tuples . . . . .	17
2.1.7	Cartesian Product . . . . .	18
2.1.8	Generalization of the Cartesian Product . . . . .	18

2.1.9	Cardinality of Set . . . . .	18
2.2	Set Operations . . . . .	18
2.2.1	Union . . . . .	18
2.2.2	Intersection . . . . .	18
2.2.3	Complement . . . . .	19
2.2.4	Symmetric Difference . . . . .	19
2.2.5	Inclusion-Exclusion Principle . . . . .	19
2.2.6	Set Identities (Morgan's Law revisited) . . . . .	19
2.3	Basics of Functions . . . . .	20
2.3.1	Definition . . . . .	20
2.3.2	Injection of a function . . . . .	20
2.3.3	Surjection of a function . . . . .	20
2.3.4	Bijection of a function . . . . .	20
2.3.5	Partial Function . . . . .	20
2.4	Relations . . . . .	20
2.4.1	Binary Relation . . . . .	20
2.4.2	Relations and Functions . . . . .	21
2.4.3	Composition of Relations . . . . .	21
2.4.4	Binary Relation on a Set . . . . .	21
2.4.5	Equivalence Relation . . . . .	22
2.4.6	Number of relation in a set . . . . .	22
2.5	Equivalence Classes . . . . .	23
2.5.1	Definition . . . . .	23
2.5.2	Example of Equivalence class . . . . .	23
2.6	Equivalence Classes and Partitions . . . . .	23
2.6.1	Equivalences with Equivalence Classes . . . . .	23
2.6.2	Partition of a Set . . . . .	23
2.6.3	Theorem between Equivalence Classes and Partitions . . . . .	24
2.6.4	Partial Ordering . . . . .	24
2.6.5	Lattice . . . . .	24
2.6.6	Partial Ordering on Cartesian Product . . . . .	25
2.6.7	Hasse Diagrams . . . . .	25
2.6.8	Comparability . . . . .	26
2.7	Sequences . . . . .	26
2.7.1	Definition . . . . .	26
2.7.2	Recurrence Relation . . . . .	27
2.8	Summations . . . . .	27
2.8.1	Summation Notation . . . . .	27
2.8.2	Geometrical Series . . . . .	27
2.9	Cardinality with Sets . . . . .	27
2.9.1	Equal Cardinality . . . . .	27
2.9.2	Countable Sets . . . . .	28
2.9.3	Infinite Countable Sets . . . . .	28
2.9.4	The set of Integers is Countable . . . . .	28
2.9.5	The Set of Rational Numbers is Countable . . . . .	28
2.9.6	The Set of Real Numbers is Uncountable . . . . .	29

<b>3</b>	<b>Algorithms</b>	<b>30</b>
3.1	Introduction and Examples . . . . .	30
3.1.1	Definition . . . . .	30
3.1.2	Binary Search . . . . .	30
3.1.3	Insertion Sort . . . . .	30
3.1.4	Greedy Algorithm . . . . .	31
3.2	Matching . . . . .	31
3.2.1	Definition . . . . .	31
3.2.2	Maximum Matching . . . . .	31
3.2.3	Stable Matching . . . . .	31
3.2.4	Preference List . . . . .	31
3.2.5	Marriage Problem . . . . .	31
3.2.6	Gale-Shapley Algorithm (proof of existence of stable maximum matching) . . . . .	31
3.3	Growth Of Functions . . . . .	33
3.3.1	Introduction . . . . .	33
3.3.2	Big-O Notation . . . . .	33
3.3.3	Examples of big-O . . . . .	33
3.3.4	Big-O Estimation for Polynomials . . . . .	33
3.3.5	Big-O Examples . . . . .	33
3.3.6	Combination of Functions . . . . .	34
3.3.7	Big-Omega Notation . . . . .	34
3.3.8	Example of Big-Omega . . . . .	35
3.3.9	Big-Theta Notation . . . . .	35
3.3.10	Big-Theta Estimation for Polynomials . . . . .	35
3.3.11	Little-o Notation . . . . .	35
3.4	Complexity of Algorithms . . . . .	35
3.4.1	Definition . . . . .	35
3.4.2	Time Complexity . . . . .	36
3.4.3	Example of Worst-case complexity (1) . . . . .	36
3.4.4	Example of Worst-case complexity (2) . . . . .	36
3.4.5	Summary of Complexity for some Algorithms . . . . .	37
3.4.6	Hierarchy of Functions . . . . .	37
<b>4</b>	<b>Induction and Recursion</b>	<b>38</b>
4.1	Mathematical Induction . . . . .	38
4.1.1	Definition . . . . .	38
4.1.2	Induction as Inference Rule . . . . .	38
4.1.3	Why Induction is Valid ? . . . . .	38
4.1.4	Example of Induction . . . . .	38
4.1.5	Example of Induction with Sets . . . . .	38
4.1.6	Strong Induction . . . . .	39
4.2	Recursive Definitions and Structural Induction . . . . .	39
4.2.1	Recursive Defined Function . . . . .	39
4.2.2	Recursively Defined Sets and Structures . . . . .	39
4.3	Recursive Algorithms . . . . .	39

4.3.1	Definition . . . . .	39
4.3.2	Recursive Algorithms Better Than Others . . . . .	40
4.3.3	Difference between Recursion and Induction . . . . .	40
<b>5</b>	<b>Number Theory and Group Theory . . . . .</b>	<b>41</b>
5.1	Division . . . . .	41
5.1.1	Division . . . . .	41
5.1.2	Properties of Division . . . . .	41
5.1.3	Division Algorithm . . . . .	42
5.2	Modular Arithmetic . . . . .	42
5.2.1	Congruence Relation . . . . .	42
5.2.2	Congruences of Sums and Products . . . . .	42
5.3	Integer Representation and Algorithm . . . . .	42
5.3.1	Representation of Integers . . . . .	42
5.3.2	Base b Representation . . . . .	42
5.3.3	Base b Expansion Algorithm . . . . .	43
5.3.4	Conversion into Basis . . . . .	43
5.4	Arithmetic with Base 2 . . . . .	43
5.4.1	Addition of Integers . . . . .	43
5.4.2	Multiplication of Integers . . . . .	44
5.5	Prime Numbers . . . . .	44
5.5.1	Definition . . . . .	44
5.5.2	Fundamental Theorem of Arithmetic . . . . .	44
5.5.3	Trial Division . . . . .	45
5.5.4	Infinity of Primes . . . . .	45
5.5.5	Distribution of Primes . . . . .	46
5.5.6	Theorem (Prime Numbers) . . . . .	46
5.5.7	Co-prime . . . . .	46
5.5.8	Theorem (Co-prime) . . . . .	46
5.5.9	Goldbach's Conjecture . . . . .	46
5.6	GCD - LCM . . . . .	46
5.6.1	Greatest Common Divisor (GCD) . . . . .	46
5.6.2	Relatively Prime . . . . .	47
5.6.3	Compute the GCD . . . . .	47
5.6.4	Complexity of the Euclidean Algorithm . . . . .	47
5.6.5	Least Common Multiple (LCM) . . . . .	47
5.6.6	Compute the LCM . . . . .	48
5.6.7	Multiplying GCD and LCM . . . . .	48
5.7	Congruence Classes . . . . .	48
5.7.1	Definition of Modulus . . . . .	48
5.7.2	Set of All Congruence Classes . . . . .	48
5.7.3	Properties of congruence classes . . . . .	48
5.7.4	Inverse and Solution Equivalence . . . . .	49
5.7.5	Theorem of Multiplicative Inverse . . . . .	49
5.7.6	Multiplicative Inverse and Primes . . . . .	49
5.7.7	Bézout Identity . . . . .	49

5.8	Commutative Groups . . . . .	50
5.8.1	Definition . . . . .	50
5.8.2	Euler's phi function . . . . .	51
5.8.3	Cartesian Product with Groups . . . . .	51
5.9	Isomorphisms . . . . .	51
5.9.1	Definition . . . . .	51
5.9.2	Implication of Isomorphism . . . . .	51
5.9.3	Definition (Order) . . . . .	53
5.9.4	Lagrange's Theorem . . . . .	54
5.9.5	Isomorphism and Orders . . . . .	54
5.9.6	Equivalence with Lagrange Theorem . . . . .	54
5.9.7	Euler's Theorem . . . . .	54
5.9.8	Corrolary to Euler (Fermat's Theorem) . . . . .	54
5.9.9	The Chinese Remainder Theorem (CRT) . . . . .	54
5.9.10	Theorem (Combine Fermat and CRT) . . . . .	55
5.9.11	Cyclic Groups . . . . .	55
5.9.12	Discrete Exponentiation/Logarithms . . . . .	55
5.10	Fields . . . . .	56
5.10.1	Lemma of Prime Numbers (Characteristic) . . . . .	56
5.10.2	Definition Isomorphism in Fields . . . . .	56
5.10.3	Interesting Facts . . . . .	56
5.11	Vector Space . . . . .	57
5.11.1	Definition . . . . .	57
5.11.2	Subspace . . . . .	57
5.11.3	Linear Independence . . . . .	57
5.11.4	Basics Theorems . . . . .	58
5.11.5	Cardinality of a vector space . . . . .	58
5.11.6	Subspace S of $F$ power $n$ . . . . .	58
5.11.7	Fulfilled a system . . . . .	58
5.11.8	Find the Dimension of a Subspace . . . . .	59
5.11.9	Important Facts . . . . .	59
5.11.10	Find the Dimension of a Subspace (Revisited) . . . . .	59
5.11.11	Cardinality and Dimension . . . . .	59
<b>6</b>	<b>Counting</b>	<b>61</b>
6.1	Basic Counting Principles . . . . .	61
6.1.1	Product Rule Principle . . . . .	61
6.1.2	Counting Functions . . . . .	61
6.1.3	The Sum Rule . . . . .	61
6.1.4	Sum Rule with Sets . . . . .	61
6.2	Pigeonhole Principle . . . . .	61
6.2.1	Definition . . . . .	61
6.2.2	Generalization of Pigeonhole Principle . . . . .	62
6.2.3	Example with Functions . . . . .	62
6.2.4	Example with Months . . . . .	62
6.3	Permutation and Combination . . . . .	62

6.3.1	Definition of Permutation . . . . .	62
6.3.2	Counting Number of Permutation . . . . .	62
6.3.3	Definition of Combination . . . . .	63
6.3.4	Example with Cards . . . . .	63
6.4	Binomial Coefficients and Identities . . . . .	63
6.4.1	Binomial Theorem . . . . .	63
6.4.2	Proof of Binomial Theorem . . . . .	63
6.4.3	Exponential Identity . . . . .	63
6.4.4	Pascal's Identity . . . . .	64
6.5	Permutation and Combination with Repetitions . . . . .	64
6.5.1	Permutation with Repetitions . . . . .	64
6.5.2	Example of r-Permutation with Repetitions . . . . .	64
6.5.3	Combination with Repetitions . . . . .	64
6.5.4	Example of r-Combination with Repetitions . . . . .	64
6.5.5	Permutations with Indistinguishable Objects . . . . .	65
<b>7</b>	<b>Advanced Counting</b>	<b>65</b>
7.1	Counting with Recurrence Relations . . . . .	65
7.1.1	Recurrence Relation . . . . .	65
7.1.2	Counting Bit Strings . . . . .	65
7.2	Solving Linear Recurrence Relations . . . . .	65
7.2.1	Linear Homogeneous Recurrence Relation . . . . .	65
7.2.2	Examples of Linear Homogeneous Recurrence Relation . .	66
7.2.3	Characteristic Equation . . . . .	66
7.2.4	Solving Linear Homogeneous Recurrence Relation of de- gree 2 . . . . .	66
7.2.5	Example of solving Linear Homogeneous Recurrence Re- lation of degree 2 . . . . .	66
7.2.6	Solving Linear Homogeneous Recurrence Relation of Ar- bitrary degree . . . . .	66
7.3	Generating Functions . . . . .	67
7.3.1	Definition . . . . .	67
7.3.2	Examples of Generating functions . . . . .	67
7.3.3	Solving Recurrence Relation with Generating Functions .	67
7.4	Counting Problem . . . . .	67
7.4.1	Extended Binomial Coefficients . . . . .	67
7.4.2	Extended Binomial Theorem . . . . .	67
7.5	Inclusion-Exclusion . . . . .	68
7.5.1	Inclusion-Exclusion Principle (Recall) . . . . .	68
7.5.2	Derangements . . . . .	68
<b>8</b>	<b>Probability</b>	<b>69</b>
8.1	Introduction to Probability . . . . .	69
8.1.1	Probability of an Event . . . . .	69
8.1.2	Probability of Complement of Events . . . . .	69
8.1.3	Probability of Union of Events . . . . .	69

8.1.4	Independence of Events . . . . .	69
8.2	Probability Theory . . . . .	70
8.2.1	Limitation of Laplace's Definition . . . . .	70
8.2.2	Probability Distribution . . . . .	70
8.2.3	Combination of Events . . . . .	70
8.2.4	Uniform Distribution . . . . .	70
8.2.5	Pairwise and Mutual Independence . . . . .	70
8.2.6	Independent Bernoulli Trials . . . . .	70
8.3	Conditional Probability . . . . .	71
8.3.1	Definition . . . . .	71
8.3.2	Bayes' Theorem . . . . .	71
8.3.3	Generalization of Bayes' Theorem . . . . .	71
<b>9</b>	<b>Advanced Probability</b>	<b>72</b>
9.1	Expected Value . . . . .	72
9.1.1	Random Variable . . . . .	72
9.1.2	Distribution of Random Variable . . . . .	72
9.1.3	Expected Value . . . . .	72
9.1.4	Expected Value for Bernoulli Trials . . . . .	72
9.1.5	Linearity of Expected Value . . . . .	73
9.1.6	Independent Random Variables . . . . .	73
9.2	Variance . . . . .	73
9.2.1	Definition . . . . .	73
9.2.2	Standard Deviation . . . . .	73
9.2.3	Example to explain the variance . . . . .	73
9.2.4	Characterization of Variance . . . . .	74
9.2.5	Variance for Bernoulli Trials . . . . .	74
9.2.6	Variance for Independent Random Variables . . . . .	74
9.3	Estimation Deviation . . . . .	74
9.3.1	Markov Inequality . . . . .	74
9.3.2	Chebyshev's Inequality . . . . .	74
9.4	Geometrical Distribution . . . . .	74
9.4.1	Definition . . . . .	74
9.4.2	Example of Geometrical Distribution . . . . .	75
<b>10</b>	<b>Entropy and Informations</b>	<b>76</b>
10.1	Symbols . . . . .	76
10.1.1	Informations and symbols . . . . .	76
10.2	Amount of Informations . . . . .	76
10.2.1	Conventions . . . . .	76
10.2.2	Hartley's Measure . . . . .	76
10.2.3	Shannon's Theory (Entropy) . . . . .	76
10.2.4	Binary Entropy Function . . . . .	77
10.3	Applications and Theorems . . . . .	77
10.3.1	Theorem 1 (Entropy Bound) . . . . .	77
10.3.2	Entropy with more than 1 random variable . . . . .	78



10.3.3	Notations . . . . .	79
10.3.4	Example with many random variables . . . . .	79
10.3.5	Theorem 2 (Addition of Entropies) . . . . .	79
10.4	Conditional Entropy . . . . .	79
10.4.1	Global definition . . . . .	79
10.4.2	Theorem 5 (Inequalities) . . . . .	80
10.4.3	Theorem 6 (Change Rule of Entropy) . . . . .	80
10.4.4	Notes About Entropy . . . . .	80
<b>11</b>	<b>Source Coding</b>	<b>81</b>
11.1	Basics of Source Coding Theory . . . . .	81
11.1.1	Schema of transmission . . . . .	81
11.1.2	Basic example . . . . .	81
11.1.3	Definitions . . . . .	81
11.1.4	Theorem 1 (First Implications) . . . . .	82
11.2	Kraft-McMillan and Uniformly Decodable Codes . . . . .	82
11.2.1	Theorem 2 (Kraft-McMillan theorem Part 1) . . . . .	82
11.2.2	Theorem 2 (Kraft-McMillan Part 2) . . . . .	82
11.2.3	Theorem 3 (Mix Implications with Theorem 1) . . . . .	82
11.3	Minimize Average Codewords Lengths . . . . .	83
11.3.1	Explanation of Average Codewords Lengths . . . . .	83
11.3.2	Theorem 4 (Lower Bound for the Average Codeword Lengths) . . . . .	83
11.4	Advanced Source Coding . . . . .	83
11.4.1	Shannon-Fano Code . . . . .	83
11.4.2	Huffman's Construction Code . . . . .	83
11.5	Generalization of Source Encoding . . . . .	84
11.5.1	Different Types of Source . . . . .	84
11.5.2	Definition of Regular Source . . . . .	84
11.5.3	Inequality of Entropy Remake for many Sequences . . . . .	84
11.5.4	Definition of Stationary Source . . . . .	85
11.5.5	Theorem 7 (Implication of Stationary Source) . . . . .	85
11.5.6	Theorem 8 (Relation between Rate and Per symbol Entropies) . . . . .	85
11.5.7	Theorem 9 (Cesàro means) . . . . .	85
11.5.8	Theorem 10 (Relation with Entropy Rate) . . . . .	86
11.5.9	Summary of Source Encoding . . . . .	86
11.5.10	Independent and Identically Distributed Symbols . . . . .	86
11.6	Binary Prefix-Free Code for Positive Integers . . . . .	86
11.6.1	Standard Method . . . . .	86
11.6.2	Elias code 1 . . . . .	86
11.6.3	Elias code 2 . . . . .	86

<b>12</b>	<b>Cryptography</b>	<b>87</b>
12.1	Privacy / Confidentiality . . . . .	87
12.1.1	Tools of cryptography . . . . .	87
12.1.2	Description of Cryptography . . . . .	87
12.1.3	Type of Attacks . . . . .	87
12.1.4	Definition of Perfect Secrecy . . . . .	88
12.1.5	Example of Perfect Secrecy: One-Time-Pad (OTP) . . . . .	88
12.1.6	Theorem (Perfect Secrecy) . . . . .	88
12.2	Public-Key Distribution . . . . .	88
12.2.1	One-Way Function . . . . .	88
12.2.2	DH-Distribution . . . . .	88
12.2.3	Paradigm Perfect Secrecy and Security . . . . .	89
12.2.4	Trapdoor One-Way Function . . . . .	89
12.3	RSA (Rivest, Shamir, Adleman) Algorithm . . . . .	89
12.3.1	Construction of the public key . . . . .	89
12.3.2	Construction of the private key . . . . .	89
12.4	Privacy for Cryptographic Systems . . . . .	89
12.4.1	Hash Function . . . . .	89
12.4.2	Digital Signature . . . . .	90
12.4.3	Trusted Agency . . . . .	90
12.4.4	Standarts . . . . .	90
12.4.5	HTTPS . . . . .	90
12.4.6	Example of IMessage (Apple) . . . . .	91
12.4.7	Complexity of computing $a^k \equiv \text{mod } m$ . . . . .	91
12.4.8	Summaty of Symmetric and Asymmetric key Paradigms . . . . .	92
<b>13</b>	<b>Channel Coding</b>	<b>93</b>
13.1	The Noisy Channel . . . . .	93
13.1.1	Erasure Channel . . . . .	93
13.1.2	Error Channel . . . . .	93
13.1.3	Detection and Correction . . . . .	93
13.2	Basics of Channel Coding . . . . .	93
13.2.1	Terminology . . . . .	93
13.2.2	The Hamming Distance . . . . .	94
13.2.3	Theorem (Hamming Distance) . . . . .	94
13.2.4	Hamming-Distance Decoder . . . . .	94
13.2.5	The Minimum Sistance of a code C . . . . .	94
13.2.6	What do we want from a Code . . . . .	94
13.2.7	Singleton Bounds . . . . .	94
13.3	Minimum Distance (MD) Decoder . . . . .	94
13.3.1	Theorem (MD-Decoder in Erasure Channel) - Sufficient . . . . .	94
13.3.2	Theorem (MD-Decoder in Error Channel) . . . . .	95
13.4	Code and Linearity . . . . .	95
13.4.1	Definition of a linear code . . . . .	95
13.4.2	Dimension of a Linear Code . . . . .	95
13.5	Hamming Weight . . . . .	95

13.5.1	Definition . . . . .	95
13.5.2	Theorem of Minimum Distance . . . . .	95
13.6	Basis and Description of Code . . . . .	96
13.6.1	Systematic Generator Matrices . . . . .	96
13.7	Decoding . . . . .	96
13.7.1	Error Pattern . . . . .	96
13.7.2	Parity Check Matrix . . . . .	97
13.7.3	Syndrome . . . . .	97
13.7.4	Theorem Minimum Distance . . . . .	97
13.7.5	Dimension of a Code with Parity Matrix . . . . .	97
13.8	Decoding based on Cosets and Syndromes . . . . .	97
13.8.1	Equivalence Relations with Codes . . . . .	97
13.8.2	Cosets . . . . .	98
13.8.3	Facts on cosets . . . . .	98
13.8.4	Application of Cosets on Code . . . . .	98
13.8.5	Standard Array . . . . .	98
13.8.6	Coset Decoder . . . . .	98
13.8.7	Theorem (Relation between Cosets and Syndromes) . . . . .	99
13.8.8	Decoding Algorithm Explanation . . . . .	99
13.8.9	Examples of Coset Decoding . . . . .	100
13.9	Reed-Solomon Codes (RS) . . . . .	101
13.9.1	Lagrange's Interpolation . . . . .	101
13.9.2	Fundamental Theorem of Algebra . . . . .	101
13.9.3	Construction . . . . .	101
13.9.4	Properties of RS Codes . . . . .	102
13.9.5	Example of RS Code . . . . .	102

# 1 Propositional Logic

## 1.1 Basic Concepts

### 1.1.1 Proposition

A proposition is a declarative sentence that is either **true** or **false**. A proposition that cannot be expressed in terms of simpler propositions are called atomic propositions.

### 1.1.2 Logical Connectives

Logical connectives are symbols that are used to connect multiple formulas. The main logical connectives are:

- The negation  $\neg$
- The inclusive conjunction  $\wedge$
- The exclusive conjunction  $\oplus$
- The disjunction  $\vee$
- The implication  $\rightarrow$
- The bi-condition  $\Leftrightarrow$

Let  $P$  be the proposition "It rains", then the negation  $\neg P$  is "it not rains".

### 1.1.3 Truth Tables

A truth table is a table that lists all possible values of a compound expression in terms of the different propositions. For example, the truth table of the expression  $P \wedge Q$  is

$P$	$Q$	$P \wedge Q$
$T$	$T$	$T$
$T$	$F$	$F$
$F$	$T$	$F$
$F$	$F$	$F$

And the main truth tables are:

$P$	$Q$	$P \vee Q$
$T$	$T$	$T$
$T$	$F$	$T$
$F$	$T$	$T$
$F$	$F$	$F$

$P$	$Q$	$P \oplus Q$
$T$	$T$	$F$
$T$	$F$	$T$
$F$	$T$	$T$
$F$	$F$	$F$

$P$	$Q$	$P \rightarrow Q$
$T$	$T$	$T$
$T$	$F$	$F$
$F$	$T$	$T$
$F$	$F$	$T$

$P$	$Q$	$P \Leftrightarrow Q$
$T$	$T$	$T$
$T$	$F$	$F$
$F$	$T$	$F$
$F$	$F$	$T$

### 1.1.4 Terminology

- A **Tautology** is a proposition always true.
- A **Contradiction** is a proposition always false.
- A **Contingency** is neither a tautology nor a contradiction.

## 1.2 Logical Equivalences

### 1.2.1 Definition

Two propositions are said **equivalent** if  $P \Leftrightarrow Q$  is a tautology. We write this like this

$$P \equiv Q \quad (1)$$

We can see equivalences in the truth tables too, by checking if the columns in a truth table giving their truth tables agree.

$P \rightarrow Q$  is **NOT** equivalent to  $Q \rightarrow P$ . But  $P \rightarrow Q \equiv \neg Q \rightarrow \neg P$

### 1.2.2 Tables of Equivalences

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

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

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

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

## 1.3 Morgan's Law

### 1.3.1 Definition

Let  $P$  and  $Q$  2 propositions. We can resume the Morgan's Law as

$$\neg(P \wedge Q) \Leftrightarrow \neg P \vee \neg Q \quad (2)$$

$$\neg(P \vee Q) \Leftrightarrow \neg P \wedge \neg Q \quad (3)$$

### 1.3.2 Proof

Let  $A = \neg(P \vee Q)$  and  $X \in A$ . By definition,  $X \notin (P \vee Q)$ . So  $X \notin P$  and  $X \notin Q \Rightarrow X \in \neg P$  and  $X \in \neg Q \Rightarrow X \in \neg P \wedge \neg Q$ . We use the same things for the second equivalence.

## 1.4 Normal Forms

### 1.4.1 Disjunctive Normal Form (DNF)

It's a disjunction of compound expressions where each compound expression is a conjunction of a set of propositional variables of their negation.

$$(P \wedge \neg Q) \vee (P \wedge R) \vee (Q \wedge T)$$

### 1.4.2 Conjunctive Normal Form (CNF)

It's a conjunction of compound expressions where each compound expression is a disjunction of a set of propositional variables of their negation.

$$(P \vee \neg Q) \wedge (P \vee R) \wedge (Q \vee T)$$

#### Theorem Of CNF

Every proposition can be put into a conjunctive normal form.

## 1.5 Universal and Existential Quantifiers

### 1.5.1 Variables

It's an object that characterize something by properties. For example, in  $Man(x) - x$  is a man,  $x$  is a variable.

### 1.5.2 Predicates

It's a statement that contains a variable and no logical connectives. The example of  $Man(x)$  is a predicate.

### 1.5.3 Predicates and Propositional Logic

Let  $M(x)$  and  $N(y)$ , 2 predicates. Then  $M(Alice) \wedge N(Bob)$  is a combination of predicates with logical operations.

### 1.5.4 Propositional Functions

We can define more general predicates defining functions like  $R(x, y) := P(x) \rightarrow P(y)$ .

### 1.5.5 Universal Quantifier

The Universal quantifier of a statement is written with the  $\forall$  symbol and it means that  $P(x)$  is true for all values  $x$  from its domain  $U$ .

### 1.5.6 Existential Quantifier

The Existential quantifier of a statement is written with the  $\exists$  symbol and it means that  $P(x)$  is true for a value  $x$  from its domain  $U$ .

## 1.6 Validity and Satisfiability

### 1.6.1 Statement Valid

A statement involving predicates and quantifiers with all variables bounded is valid if it's true for all domains and every propositional function substituted for the predicates in the assertion.

### 1.6.2 Statement Satisfiable

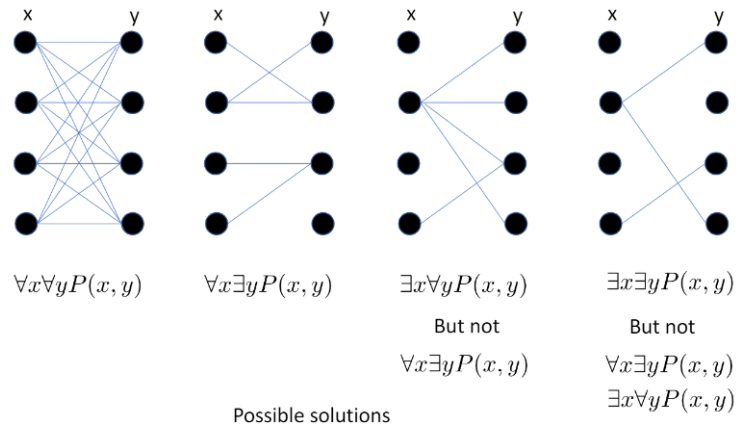
A statement involving predicates and quantifiers with all variables bounded is valid if it's true for some domains. Otherwise it's unsatisfied.

## 1.7 The Morgan's Law for Quantifiers

<i>Negation</i>	<i>Equivalent Statement</i>
$\neg \exists x P(x)$	$\forall x \neg P(x)$
$\neg \forall x P(x)$	$\exists x \neg P(x)$

Warning, the ordering of quantifiers is **critical**.

## Visualization: x,y connected if P(x,y) true



### 1.8 Visualization of Quantifiers

### 1.9 Proofs

#### 1.9.1 Argument

An argument is a sequence of propositions. Every proposition without the final are called premises and the last the conclusion.

The argument is said valid if and only if the premises imply the conclusion.

#### 1.9.2 Inference Rules

An inference rule is a simple argument form that will be used to construct more complex argument forms. The lines on the fraction bar are propositions and there are put together with AND logic. The proposition under the fraction bar is the implication of the premises over the fraction bar.

<i>Rules</i>	<i>Name</i>	<i>Rules</i>	<i>Name</i>
$\frac{P}{P \vee Q}$	<i>Addition</i>	$\frac{P \wedge Q}{P}$	<i>Simplification</i>
$\frac{P \vee Q \text{ and } \neg P}{Q}$	<i>Disjunctive Syllogism</i>	$\frac{P \rightarrow Q \text{ and } P}{Q}$	<i>Modus Ponens</i>



# Inference Rule: Modus Tollens

$$\frac{p \rightarrow q \quad \neg q}{\therefore \neg p}$$

**Corresponding Tautology:**

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

## 1.9.3 Inference Rules with Quantifiers

An inference rule with quantifiers is an inference rule but we introduced quantifiers in it.

<i>Rules</i>
$\frac{\forall x P(x)}{P(c)}$
$\frac{\exists x P(x)}{P(c) \text{ for some element } c}$

## 1.9.4 Type of Proofs

- **Direct proof:** Assume  $P$  is true, then we use rules of inference, axioms till the statement  $Q$  result.
- **Proof by Contraposition:** Assume  $\neg Q$  is true, then we use rules of inference, axioms till the statement  $\neg P$  result. Since  $P \rightarrow Q \equiv \neg Q \rightarrow \neg P$
- **Proof by Contradiction:** Assume  $P$  and  $\neg Q$  are true, since  $P \rightarrow Q \equiv (P \wedge \neg Q) \rightarrow F$

## 2 Basic Structures: Sets, Functions, Sequences and Sums

### 2.1 Basics of Sets

#### 2.1.1 Definition

A set is an unordered collection of distinct objects (for example numbers).

Sets can be elements of sets:  $\{\{a, b, c\}, d\}$  and  $\emptyset \neq \{\emptyset\}$

#### 2.1.2 Set Equality

Let  $A$  and  $B$ , 2 sets.  $A$  and  $B$  are equal if and only if  $A$  and  $B$  have the same elements.

$$\forall x(x \in A \longleftrightarrow x \in B) \quad (4)$$

#### 2.1.3 Subset

The set  $A$  is a subset of the set  $B$  if and only if every element of  $A$  is also in  $B$ . We write  $A \subseteq B$ .

$$\forall x(x \in A \rightarrow x \in B) \quad (5)$$

Because  $a \in \emptyset$  is always false, the empty set  $\emptyset$  is a subset for all subset  $S$ .

#### 2.1.4 Proper Subset

Let  $A$  and  $B$  be 2 sets and let  $A \subseteq B$  but  $A \neq B$ , then we say that  $A$  is a proper subset of  $B$ , and it's written  $A \subset B$ , if and only if

$$\forall x(x \in A \rightarrow x \in B) \wedge \exists x(x \in B \wedge x \notin A) \quad (6)$$

#### 2.1.5 Power Set

We define the power set, denoted as  $\mathcal{P}(A)$ , as the set of all subsets of a set  $A$ . if  $A = \{a, b\}$  then  $\mathcal{P}(A) = \{\emptyset, \{a\}, \{b\}, \{a, b\}\}$ .

#### 2.1.6 Tuples and Equality of Tuples

The ordered  $n$ -tuples  $(a_1, \dots, a_n)$  is the ordered collection that has  $a_1$  as it's first element and  $a_n$  for it's last. 2  $n$ -tuples are equal if and only if their corresponding elements are equal.

2-tuples are called **ordered pairs**.

### 2.1.7 Cartesian Product

Let  $A$  and  $B$ , 2 sets. We define the Cartesian product of  $A$  and  $B$  as the set of ordered pairs  $(a, b)$  where  $a \in A$  and  $b \in B$ , and denoted as  $A \times B$

$$\mathcal{A} \times \mathcal{B} = \{(a, b) | a \in A \wedge b \in B\} \quad (7)$$

A subset  $\mathcal{R}$  of the Cartesian product  $\mathcal{A} \times \mathcal{B}$  is called a **relation** from the set  $\mathcal{A}$  to the set  $\mathcal{B}$ .

Note that in general  $\mathcal{A} \times \mathcal{B} \neq \mathcal{B} \times \mathcal{A}$

### 2.1.8 Generalization of the Cartesian Product

More generally, we define the Cartesian product of the sets  $A_1, \dots, A_n$ , denoted by  $A_1 \times \dots \times A_n$ , as the set of ordered  $n$ -tuples  $(a_1, \dots, a_n)$  where  $a_i$  belongs to  $A_i$  for  $i = 1, \dots, n$ .

$$A_1 \times \dots \times A_n = \{(a_1, \dots, a_n) | a_i \in A_i \text{ for } i = 1, \dots, n\} \quad (8)$$

### 2.1.9 Cardinality of Set

The cardinality of a set  $A$ , denoted as  $|A|$ , is the number of distinct elements in  $A$ . We say that the set is finite if there are exactly  $n$  non-negative elements ; otherwise it's infinite.

The cardinality of the Power set  $\mathcal{P}(A) = 2^n$  where  $n$  is the number of elements in  $A$ . Also, if  $|A| = n$  and  $|B| = m \Rightarrow |A \times B| = nm$ .

## 2.2 Set Operations

### 2.2.1 Union

Let  $A$  and  $B$ , 2 sets. The union of  $A$  and  $B$ , denoted by  $A \cup B$ , is the set:

$$\{x | x \in A \vee x \in B\} \quad (9)$$

### 2.2.2 Intersection

Let  $A$  and  $B$ , 2 sets. The intersection of  $A$  and  $B$ , denoted by  $A \cap B$ , is the set:

$$\{x | x \in A \wedge x \in B\} \quad (10)$$

Sets that have no intersection ( $A \cap B = \emptyset$ ) are called **disjoint**

### 2.2.3 Complement

Let  $A$  be a set. The complement of  $A$  with respect to the universe set  $\mathcal{U}$ , denoted by  $\bar{A} = \mathcal{U} - A$ , is the set:

$$\{x \in \mathcal{U} | x \notin A\} \quad (11)$$

### 2.2.4 Symmetric Difference

Let  $A$  and  $B$ , 2 sets. The symmetric difference of  $A$  and  $B$ , denoted by  $A \oplus B$ , is the set:

$$(A - B) \vee (B - A) \quad (12)$$

### 2.2.5 Inclusion-Exclusion Principle

The inclusion-exclusion principle is a principle that represent a counting technique which generalizes the familiar method of obtaining the number of elements in the union of two finite sets.

$$|A \cup B| = |A| + |B| - |A \cap B| \quad (13)$$

#### Cardinality of Sets Union

Let  $A_1, \dots, A_n$  be  $n$  finite sets. then  $|A_1 \cup \dots \cup A_n| = \sum_i^n |A_i| - \sum_{1 \leq i < k \leq n} |A_i \cap A_k| + \sum_i^n |A_i| - \sum_{1 \leq i < j < k \leq n} |A_i \cap A_k \cap A_j| + \dots + (-1)^{n+1} |\bigcap_i^n A_i|$

### 2.2.6 Set Identities (Morgan's Law revisited)

Let  $A$  and  $B$ , 2 sets. Then

$$\overline{A \cap B} = \bar{A} \cup \bar{B} \quad (14)$$

$$\overline{A \cup B} = \bar{A} \cap \bar{B} \quad (15)$$

Let  $\overline{A \cap B} = \{x | x \notin A \cap B\} = \{x | \neg(x \in A \cap B)\} = \{x | \neg(x \in A \wedge x \in B)\} = \{x | \neg x \in A \vee \neg x \in B\} = \{x | x \in \bar{A} \vee x \in \bar{B}\} = \{x | x \in \bar{A} \cup \bar{B}\} = \bar{A} \cup \bar{B}$

## 2.3 Basics of Functions

### 2.3.1 Definition

A function  $f$  from the set  $A$  to the set  $B$  is an assignment of exactly one element of  $B$  to each element of  $A$ .

$$f : A \rightarrow B \quad (16)$$

$$x \mapsto f(x) \quad (17)$$

We define  $A$  as the domain  $D(f)$  and  $B$  the image of the co-domain. If  $f(a) = b$ , then  $a$  is the pre-image of  $b$  and  $b$  is the image of  $a$  by  $f$ .

### 2.3.2 Injection of a function

We say that  $f$  is injective, or one-to-one, if and only if

$$f(a) = f(b) \text{ implies that } a = b \text{ and } \forall a, b \in D(f) \quad (18)$$

### 2.3.3 Surjection of a function

We say that  $f$  is surjective, or onto, if and only if

$$\forall b \in B : \exists a \in A : f(a) = b \quad (19)$$

### 2.3.4 Bijection of a function

We say that  $f$  is bijective if and only if the function is one-to-one and onto. With this, we can find the **inversion of the function**  $f^{-1}(x)$

$$f^{-1}(y) = x \Leftrightarrow f(x) = y \quad (20)$$

### 2.3.5 Partial Function

A partial function from the set  $A$  to the set  $B$  is an assignment to each element  $a$  in a subset of  $A$ , called the domain of definition of  $f$ , of a unique element  $b \in B$ . For example, the function  $f : \mathbb{Z} \rightarrow \mathbb{R}$  with  $f(x) = \sqrt{x}$  is a partial function because it's only the positive integers that can be sent under the sqrt, and the positive integers  $\mathbb{N}$  are a subset of the domain  $\mathbb{Z}$ . The definition is fulfilled.

## 2.4 Relations

### 2.4.1 Binary Relation

A binary relation  $R$  from a set  $A$  to  $B$  is a subset of the cartesian product of  $A$  and  $B$ :  $R \subseteq A \times B$ .

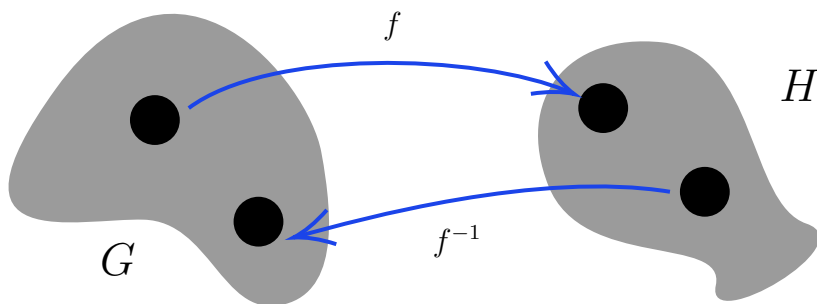
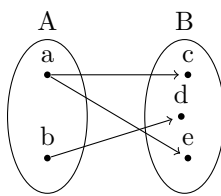


Figure 1: Bijection of a function



#### 2.4.2 Relations and Functions

A function  $f : A \rightarrow B$  can also be defined as a subset of  $A \times B$ , like a relation. A function  $f$  from  $A$  to  $B$  contains one, and only one ordered pair  $(a, b)$  for every element  $a \in A$ .

$$\forall x[x \in A \rightarrow \exists y[y \in B \wedge (x, y) \in f]] \quad (21)$$

$$\forall x, y_1, y_2[(x, y_1) \in f \wedge (x, y_2) \in f] \rightarrow y_1 = y_2 \quad (22)$$

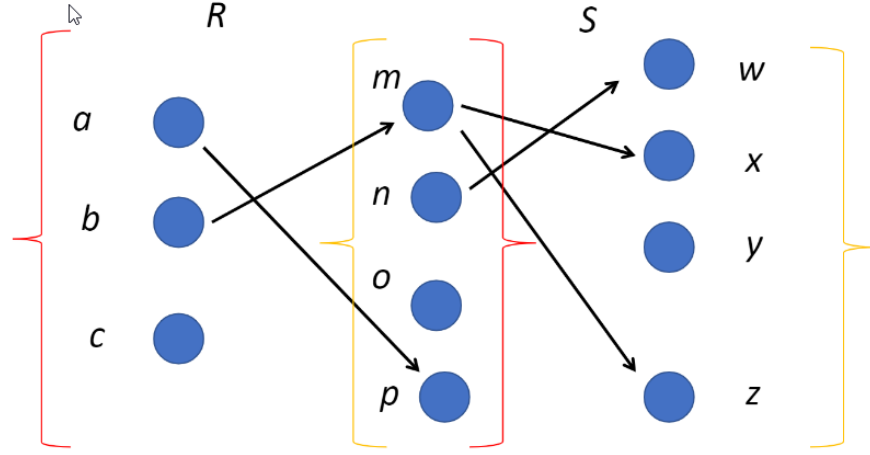
Relations are more general than functions !

#### 2.4.3 Composition of Relations

Let  $R$  be a relation from the set  $A$  to the set  $B$  and let  $S$  be a relation from the set  $C$  to the set  $D$ . The **composite** of  $R$  and  $S$  is the relation consisting of ordered pairs  $(a, c)$ , where  $a \in A$  and  $c \in C$ , and for which there exists an element  $b \in B$  such that  $(a, b) \in R$  and  $(b, c) \in S$ . We denote the composition of  $R$  and  $S$  like this  $R \circ S$

#### 2.4.4 Binary Relation on a Set

A binary relation on a set  $A$  is denoted by  $R$  and it's a subset of  $A \times A$ .



$$S \circ R = \{(b, x), (b, z)\}$$

Figure 2: Example of composition of relations

#### 2.4.5 Equivalence Relation

Let  $A$  be a set. We say that if a relation is symmetric, transitive and reflexive, then it's an **equivalence relation**. 2 elements  $a$  and  $b$  that are related by an equivalence are called **equivalent** and they are written  $a \sim b$ .

- $R$  is reflexive if and only if  $(a, a) \in R \forall a \in A$ . Reflexive:  $\forall x(x \in A \rightarrow (x, x) \in R)$
- $R$  is symmetric if and only if  $(b, a) \in R$  whenever  $(a, b) \in R$ . Symmetric:  $\forall x, y((x, y) \in R \rightarrow (y, x) \in R)$
- $R$  is transitive if whenever  $(a, b) \in R$  and  $(b, c) \in R$  then  $(a, c) \in R$ . Transitive:  $\forall x, y, z((x, y) \in R \wedge (y, z) \in R \rightarrow (x, z) \in R)$
- $R$  is anti-symmetric if and only if  $\forall a, b \in A$  if  $(a, b) \in R$  and  $(b, a) \in R$ , then  $a = b$ . Anti-Symmetric:  $\forall x, y((x, y) \in R \wedge (y, x) \in R \rightarrow x = y)$

#### 2.4.6 Number of relation in a set

Let  $A$  be a set. We know that  $A \times A$  has  $|A|^2$  elements when  $A$  has  $|A|$  elements. This implies

Number of relation in a set

The number of relations in a set  $A = 2^{|A|^2}$

## 2.5 Equivalence Classes

### 2.5.1 Definition

Let  $R$  be an equivalence relation on a set  $A$ . The set of all elements that are related to an element  $a$  of  $A$  is called an equivalence class of  $a$ . The equivalence class of  $a$  with respect to  $R$  is denoted by  $[a]_R$ . If  $b \in [a]_R$ , then  $b$  is called the representative of this equivalence class.

### 2.5.2 Example of Equivalence class

Let  $R = \{(a, b) \in \mathbb{R} \times \mathbb{R} \mid a - b \in \mathbb{Z}\}$ . What is the equivalence class of the element 0 ?  $[0]_R \Rightarrow 0 - b \in \mathbb{Z} \Rightarrow -b \in \mathbb{Z} \Rightarrow b \in \mathbb{Z}$ . So in conclusion, we can say that  $[0]_R = \mathbb{Z}$ .

## 2.6 Equivalence Classes and Partitions

### 2.6.1 Equivalences with Equivalence Classes

Equivalences with equivalence classes

Let  $R$  be an equivalence relation. These statements for elements  $a$  and  $b$  of the set  $A$  are equivalent

- $R(a, b)$
- $[a] = [b]$
- $[a] \cap [b] \neq \emptyset$

### 2.6.2 Partition of a Set

A partition of a set  $S$  is a collection of disjoint non-empty subsets of  $S$  that have  $S$  as their union. Formally for an index set  $I$ , the collection of subsets  $A_i$ , where  $i \in I$  forms a partition of  $S$  if and only if

- $A_i \neq \emptyset \forall i \in I$
- $A_i \cap A_j = \emptyset$  if  $i \neq j$
- $\bigcup_{i \in I} A_i = S$



### 2.6.3 Theorem between Equivalence Classes and Partitions

#### Union of equivalence classes

Let  $R$  be an equivalence relation. Then the equivalence classes of  $R$  form a partition of  $S$ . Conversely, given a partition on  $\{A_i | i \in I\}$  of a set  $S$ , there is an equivalence relation on  $R$  that has the sets  $A_i$  as it's equivalence classes.

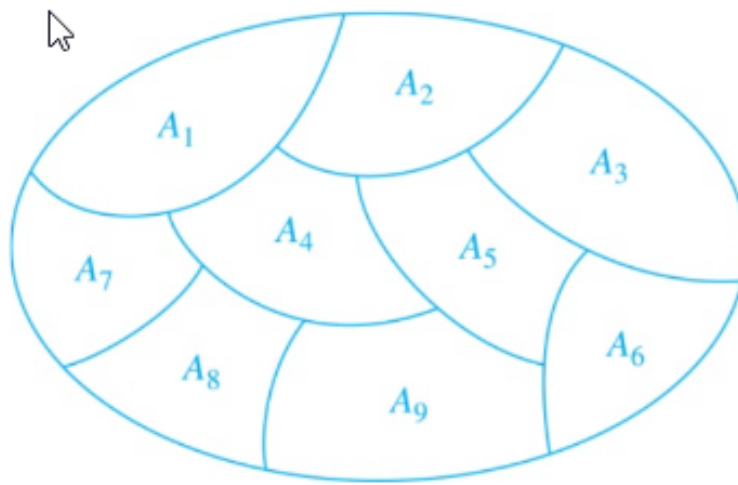


Figure 3: Example of partition of a set

### 2.6.4 Partial Ordering

A relation  $R$  on a set  $S$  is called a partial ordering if it's reflexive, anti-symmetric and transitive. A set together with a partial ordering  $R$  is called a **partially ordered set** or **poset** denoted by  $(S, R)$ . Some examples of posets are  $(\mathbb{Z}, \geq)$ ,  $(\mathbb{Z}^+, |)$  or  $(\mathcal{P}(S), \subseteq)$ .

### 2.6.5 Lattice

A poset in which every pair of elements has both a least upper bound and a greatest lower bound is called a lattice.

### 2.6.6 Partial Ordering on Cartesian Product

Given two posets  $(A_1, \prec_1)$   $(A_2, \prec_2)$ , the **lexicographic ordering on**  $A_1 \times A_2$  is defined by specifying that  $(a_1, a_2)$  is less than  $(b_1, b_2)$  that is  $(a_1, a_2) \prec (b_1, b_2)$ , either if  $a_1 \prec_1 b_1$  or if  $a_1 = b_1$  and  $a_2 \prec_2 b_2$ .

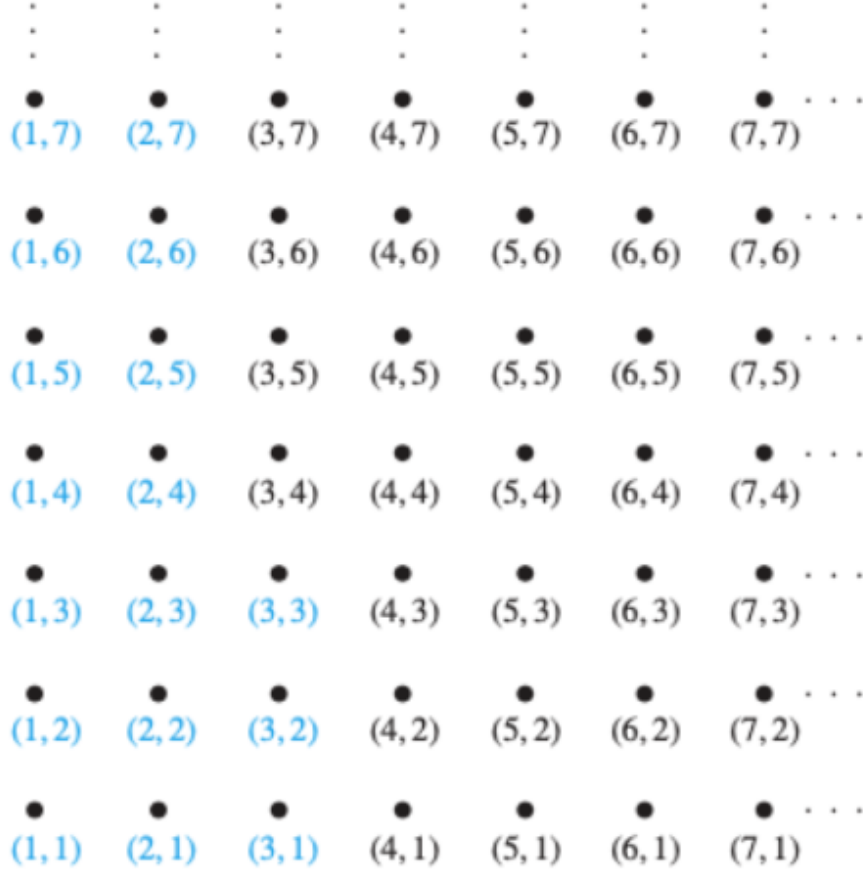


Figure 4: All ordered pairs less than  $(3,4)$

### 2.6.7 Hasse Diagrams

If a relation is reflexive and transitive, the representation as directed graph can be simplified. If  $R$  is a partial order then we can (a) omit self-loops, (b) omit transitive edges and (c) assume that arrows point upwards.

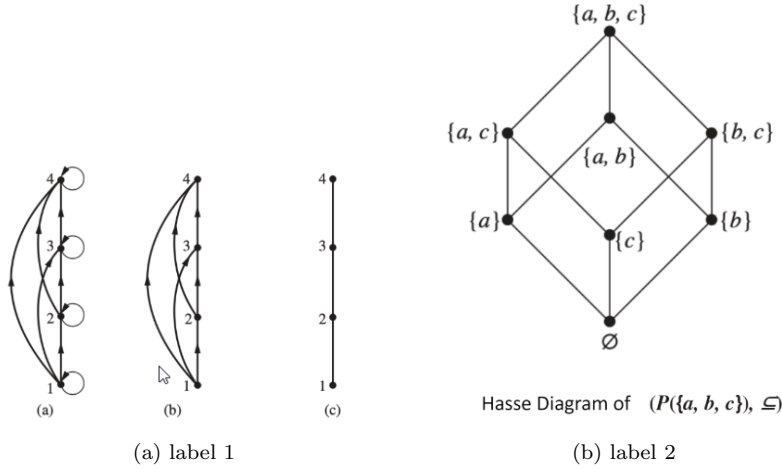


Figure 5: Representation of 2 Hasse Diagrams

### 2.6.8 Comparability

The symbol  $\preceq$  is used to denote the relation in any poset.

- **Definition 2:** The elements  $a$  and  $b$  of a poset  $(S, \preceq)$  are **comparable** if either  $a \preceq b$  or  $b \preceq a$ . When  $a$  and  $b$  are elements of  $S$  so that neither  $a \preceq b$  or  $b \preceq a$ , then  $a$  and  $b$  are **incomparable**.
- **Definition 3:** If  $(S, \preceq)$  is a poset and every 2 elements of  $S$  are comparable,  $S$  is a **totally ordered set** and  $\preceq$  is called a **total order**.
- **Definition 4:**  $(S, \preceq)$  is a **well-ordered set** if it's a poset such that  $\preceq$  is a total ordering and every non-empty subset of  $S$  has a least element.

For example,  $(\mathbb{Z}, \leq)$  is totally ordered because with  $a, b \Rightarrow a \leq b, b \leq b$  (or both).  $(\mathbb{Z}^+, |)$  is not totally ordered because for example the element  $(5, 7)$ , 5 not divide 7 and 7 not divide 5.

## 2.7 Sequences

### 2.7.1 Definition

A sequence is a function from a subset of the Integers to a set  $S$ . Usually it's either the set  $\mathbb{Z}^+$  or  $\mathbb{N}$ . We write  $a_n$  to denote the image  $f(n)$  of  $n$ . We can have different progressions:

- **Arithmetic Progression:**  $a, a+d, a+2d, \dots$  gives us the sequence  $f(n) = a + nd$
- **Geometrical Progression:**  $a, ar, ar^2, \dots$  gives us the sequence  $f(n) = ar^n$

### 2.7.2 Recurrence Relation

A recurrence relation for the sequence  $a_n$  is an expression that express  $a_n$  in terms of a finite number  $k$  of the preceding terms of the sequence. For example

$$a_n = f(a_{n-1}, \dots, a_{n-k}) \quad (23)$$

## 2.8 Summations

### 2.8.1 Summation Notation

Given a sequence  $a_n$ , the notation

$$\sum_{j=m}^n a_j \quad (24)$$

denotes the sum of the terms  $a_m, a_{m+1}, \dots, a_n$

### 2.8.2 Geometrical Series

if  $a$  and  $r$  are real numbers and  $r \neq 0$ , then

$$\sum_{j=0}^n ar^j = \begin{cases} \frac{ar^{n+1}-a}{r-1} & \text{if } r \neq 1 \\ (n+1)a & \text{if } r = 1 \end{cases} \quad (25)$$

Proof:

Let  $S_n = \sum_{j=0}^n ar^j \Leftrightarrow rS_n = r \sum_{j=0}^n ar^j = \sum_{j=0}^n ar^{j+1} = \sum_{k=1}^{n+1} ar^k = (\sum_{k=0}^n ar^k) + (ar^{n+1} - a) = S_n + (ar^{n+1} - a)$ . From this equalities, we see that  $rS_n = S_n + (ar^{n+1} - a)$ . Solving for  $S_n$  shows that if  $r \neq 1$ , then  $S_n = \frac{ar^{n+1}-a}{r-1}$ . if  $r = 1$  then  $S_n = \sum_{j=0}^n ar^j = \sum_{j=0}^n a = (n+1)a$

## 2.9 Cardinality with Sets

### 2.9.1 Equal Cardinality

Value of Cardinality between 2 sets

Let  $A$  and  $B$ , 2 sets. Then

- $|A| = |B| \Leftrightarrow$  there is a bijection between  $A$  and  $B$
- If there is an injection from  $A$  to  $B$ , then  $|A| \leq |B|$

### 2.9.2 Countable Sets

A set that is either finite or has the same cardinality of  $\mathbb{Z}^+$  is called **countable**. If it's not, then it's **uncountable**.

#### Countable Sets

A set  $S$  is countable if and only if we can list the elements in a sequence indexed, so there exist a bijection  $\mathbb{Z}^+ \rightarrow S$

### 2.9.3 Infinite Countable Sets

When an infinite set is countable, its cardinality is  $\aleph_0$ , said aleph null. But how is it possible? The great example of this is the Hilbert Hotel, a hotel with infinite rooms. If the hotel is full and a new guest arrives, we move all guests in the room number  $n + 1$ . With this, we can put the new guest in the first room. In this set, we can count the number of rooms, there is a bijection between the positive integers and the rooms. Even if there is an infinite number of new guests, we can say that the guest in room 1 goes to room 2, the guest in room 2 in the room 4, the 3 into the 6,... and with this, we have an infinite new rooms opened.

### 2.9.4 The set of Integers is Countable

We define a bijection  $f : \mathbb{Z}^+ \rightarrow \mathbb{Z}$  with  $f(n) = \frac{n}{2}$  if  $n$  is even and  $f(n) = \frac{-(n+1)}{2}$  if  $n$  is odd. With this, every element of  $\mathbb{Z}$  is related with an element of  $\mathbb{Z}^+$ .

### 2.9.5 The Set of Rational Numbers is Countable

Let us define the mapping  $f : \mathbb{Q} \rightarrow \mathbb{Z} \times \mathbb{N}$  like this:

$$\forall \frac{p}{q} \in \mathbb{Q} : f\left(\frac{p}{q}\right) = (p, q) \quad (26)$$

where  $\frac{p}{q}$  is in canonical form. Then  $f$  is clearly injective. And from the fact that a cartesian product between 2 countable sets is countable, then we have that  $\mathbb{Z} \times \mathbb{N}$  is countable infinite.

### 2.9.6 The Set of Real Numbers is Uncountable

The proof is based on the Cantor Diagonalization. We will prove that the interval  $[0, 1[ \subset \mathbb{R}$  is uncountable. Let us write some numbers from this interval.

$$\begin{aligned} r_1 &= 0.046436 & r_2 &= 0.354367 & r_3 &= 0.014054 & r_4 &= 0.463256 \\ r_5 &= 0.963621 & \dots & r_n &= 0.r_{n1}r_{n2}\dots \end{aligned}$$

Now let us create the number  $x \in [0, 1[$  formed with the  $k$ -position of the numbers written above. Here we take the first 1 number after the dot of the  $r_1$ , so 0. Then we take the second number after the dot of  $r_2$ , and so on. And now we say that if the number is 4, we keep 4 and otherwise, we put a 3. We create here the number  $x = 0.3433 \in [0, 1[$ . But here  $x$  is clearly not in the sequence formed above. he cannot be  $r_1$  because the first number after the dot is different, same for the second, the third number,... We prove that we create a new number and so we cannot create a bijection from the positive integers !

## 3 Algorithms

### 3.1 Introduction and Examples

#### 3.1.1 Definition

An algorithm is a finite set of well-defined, computer-implementable instructions to perform a specified task. Algorithms can be expressed in pseudo-code.

```
procedure max(a_1,...,a_n : integers)
  tmp_max := a_1
  for(i := 2 to n)
    if(tmp_max < a_i) then tmp_max := a_i

  return tmp_max
```

#### 3.1.2 Binary Search

It's a simple algorithm of search for an element in a list of numbers in increasing order that works with the fact that at each step, we divide the set of research in 2 until we find what we want. With numbers in increasing order, let us define that we want the number 12 in the interval  $[1, 20]$ . First step, we take the middle of the interval, so 10. 12 is greater than 10, so we search in the subset  $[10, 20]$ . Now we take the middle of this interval, so 15. 12 is less than 15 so now we take the subset  $[10, 13]$ . And so on until we find the good number.

#### 3.1.3 Insertion Sort

It's a good algorithm for sorting a list of numbers. The algorithm begins with the second element and compare it with the first element. If the second is lower than the first, it inverses the 2 elements, otherwise it does nothing. Next, the third element is compared with the 2 first elements and put where it has to be and so on.

```
procedure insertion_sort(a_1,...,a_n : real numbers with n >= 2)
  for(j := 2 to n)
    i := 1
    while(a_j > a_i and i < j)
      i := i + 1
    m := a_j
    for(k := 0 to j - i - 1)
      a_{j-k} := a_{j-k-1}
    a_i := m
```

### 3.1.4 Greedy Algorithm

A greedy algorithm will take the "best choice" at each step but it won't always produce an optimal solution.

## 3.2 Matching

### 3.2.1 Definition

Given a finite set  $A$ , a matching of  $A$  is a set of (unordered) pairs of distinct elements of  $A$  where any element occurs in at most one pair. For example,  $(\{(1, 2), (3, 4)\})$  is a matching but not  $(\{(2, 2), (2, 4)\})$ .

### 3.2.2 Maximum Matching

A maximum matching is a matching that contains the largest possible number of pair. For example  $(\{(1, 2), (3, 4)\})$  with the elements  $\{1, 2, 3, 4\}$ .

### 3.2.3 Stable Matching

Task: Pair elements from equally sized two groups considering their preferences for members of the other group so that there are no ways to improve the preferences. This notion is not really hard to understand. A stable matching is the fact that pairs of elements from 2 sets with the same cardinality are stable, so no element from a pair wants to be more with an other element of an other pair than the element with him. If we have the pairs  $(a, b)$  and  $(c, d)$ , and  $a$  likes more  $c$  than  $b$ , and  $c$  likes more  $a$  than  $d$ , we have an unstable matching...

### 3.2.4 Preference List

A preference list  $L_x$  defines for every element  $x \in A$  the order  $m$  in which the element prefers to be paired with.  $x \in A$  prefers  $y$  to  $z$  if  $y$  precedes  $z$  on  $L_x$ .

### 3.2.5 Marriage Problem

Given a set with even cardinality, partition  $A$  into two disjoint subsets  $A_1$  and  $A_2$  with  $A_1 \cup A_2 = A$  and  $|A_1| = |A_2|$ . A **matching** is a bijection from the elements of one set to the elements of the other set. That means that pairs can only consist of one element of  $A_1$  and  $A_2$  each.

### 3.2.6 Gale-Shapley Algorithm (proof of existence of stable maximum matching)

Let  $M$  be the set of pairs under construction. Initially,  $M = \emptyset$



```

while M < |A_1|:
    Select an unpaired x in A_1
    Let x propose to the first element y in A_2 on L_x:

    if(y is unpaired)
        Add (x,y) to M
    else
        Let z in A_1 be the element that y is paired with
        to (ie. (y,z) in M)
        if(z precedes x on L_x)
            Remove y from L_x
        else
            Replace (y,z) in M by (x,y) and remove y
            from L_x

```

**Examples** Assume that  $k = 4$ , that  $A_1 = \{x_1, x_2, x_3, x_4\}$ ,  $A_2 = \{y_1, y_2, y_3, y_4\}$ , that all  $x \in A_1$  have the same preference list  $L_x = (y_1, y_2, y_3, y_4)$ , and that  $L_{y_1} = (x_2, x_1, x_3, x_4)$ ,  $L_{y_2} = (x_4, x_3, x_2, x_1)$ ,  $L_{y_3} = (x_2, x_3, x_4, x_1)$  and  $L_{y_4} = (x_4, x_1, x_2, x_3)$ . Assume furthermore that the unmatched  $x \in A_1$  with the lowest index is always selected (why does this not make a difference?). During the construction the following steps are performed :

```

x1 proposes y1: y1 accepts x1's proposal
M: (x1,y1)
x2 proposes y1: y1 dumps current x1 and accepts x2's proposal
M: (x2,y1)
x1 proposes y2: y2 accepts x1's proposal
M: (x1,y2) (x2,y1)
x3 proposes y1 but y1 prefers current x2 to x3
M: (x1,y2) (x2,y1)
x3 proposes y2: y2 dumps current x1 and accepts x3's proposal
M: (x2,y1) (x3,y2)
x1 proposes y3: y3 accepts x1's proposal
M: (x1,y3) (x2,y1) (x3,y2)
x4 proposes y1 but y1 prefers current x2 to x4
M: (x1,y3) (x2,y1) (x3,y2)
x4 proposes y2: y2 dumps current x3 and accepts x4's proposal
M: (x1,y3) (x2,y1) (x4,y2)
x3 proposes y3: y3 dumps current x1 and accepts x3's proposal
M: (x2,y1) (x3,y3) (x4,y2)
x1 proposes y4: y4 accepts x1's proposal
M: (x1,y4) (x2,y1) (x3,y3) (x4,y2)
resulting stable maximum matching: (x1,y4) (x2,y1) (x3,y3) (x4,y2)

```

Figure 6: Gale-Shapley Algorithm

### 3.3 Growth Of Functions

#### 3.3.1 Introduction

In computer science, we want to describe the way that a function grows. We want to characterize the efficiency of algorithms by this growth of functions.

#### 3.3.2 Big-O Notation

The big-O notation is used to describe the limiting behaviour of a function when the arguments tend towards a particular value or infinity. It describes the upper limit. We say that  $f(x)$  is  $\mathcal{O}(g(x))$  if there are constants  $C$  and  $k$  such that

$$|f(x)| \leq C|g(x)| \quad \forall x > k \quad (27)$$

$C$  and  $k$  are called **witnesses**.

#### 3.3.3 Examples of big-O

$f(x) = x^2 + 2x + 1$  is  $\mathcal{O}(x^2)$

If  $x > 1$  then  $x < x^2$  and  $1 < x^2$ . Therefore

$$0 \leq f(x) = x^2 + 2x + 1 \leq x^2 + 2x^2 + x^2 = 4x^2 \quad (28)$$

We choose  $C = 4$  and  $k = 1$ .

#### 3.3.4 Big-O Estimation for Polynomials

##### Complexity of Polynomials

Let  $p(x) = a_n x^n + a_{n-1} x^{n-1} + \dots + a_1 x + a_0$  be a polynomial with  $a_i \in \mathbb{R}$  and  $a_n \neq 0$ . Then  $p(x)$  is  $\mathcal{O}(x^n)$

#### 3.3.5 Big-O Examples

- $\forall u > v \in \mathbb{R}$ ,  $n^v$  is  $\mathcal{O}(n^u)$  but  $n^u$  is not  $\mathcal{O}(n^v)$
- $\forall a > 0, b > 0, u > v \in \mathbb{R}$ ,  $\log_b(n^v)$  is  $\mathcal{O}(\log_b(n^u))$  and  $\log_b(n^u)$  is  $\mathcal{O}(\log_b(n^v))$
- $n!$  is  $\mathcal{O}(n^n)$
- $\log(n!)$  is  $\mathcal{O}(\log(n))$

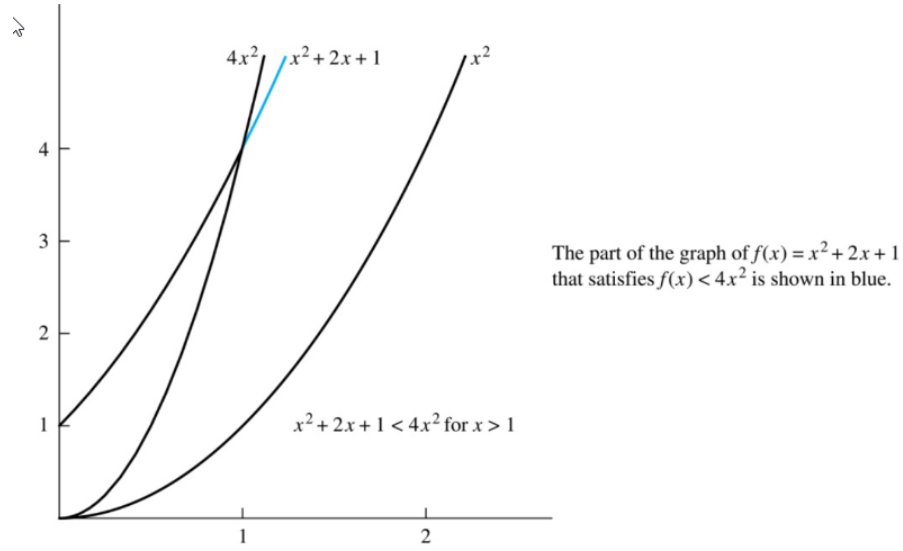


Figure 7: Graph of the complexity of the function  $f(x)$

### 3.3.6 Combination of Functions

- if  $f(x)$  is  $\mathcal{O}(g(x))$  and  $g(x)$  is  $\mathcal{O}(h(x)) \Rightarrow f(x)$  is  $\mathcal{O}(h(x))$
- If  $f_1(x)$  is  $\mathcal{O}(g_1(x))$  and  $f_2(x)$  is  $\mathcal{O}(g_2(x)) \Rightarrow (f_1(x)f_2(x))$  is  $\mathcal{O}(g_1(x)g_2(x))$
- If  $f_1(x)$  and  $f_2(x)$  are  $\mathcal{O}(g(x)) \Rightarrow (f_1(x) + f_2(x))$  is  $\mathcal{O}(g(x))$
- If  $f_1(x)$  is  $\mathcal{O}(g_1(x))$  and  $f_2(x)$  is  $\mathcal{O}(g_2(x)) \Rightarrow (f_1(x) + f_2(x))$  is  $\mathcal{O}(\max(g_1(x), g_2(x)))$

### 3.3.7 Big-Omega Notation

The big- $\Omega$  notation is used to describe the limiting behaviour of a function when the arguments tend towards a particular value or infinity. It not describe the upper limit but the lower limit. We say that  $f(x)$  is  $\Omega(g(x))$  if there are constants  $C$  and  $k$  such that

$$|f(x)| \geq C|g(x)| \quad \forall x > k \quad (29)$$

$C$  and  $k$  are called **witnesses**

Equivalence Big-O and Big-Omega

$$f(x) \text{ is } \Omega(g(x)) \Leftrightarrow g(x) \text{ is } \mathcal{O}(f(x))$$

### 3.3.8 Example of Big-Omega

The function  $f(x) = x^3 + 8x^2 + 7$  is  $\Omega(x^3)$  because  $g(x) = x^3$  is  $\mathcal{O}(x^3 + 8x^2 + 7)$

### 3.3.9 Big-Theta Notation

The big- $\Theta$  notation is used to describe a function that is  $\mathcal{O}$  something and  $\Omega$  something. By definition we describe the Theta notation by this

$$C_1|g(x)| \leq |f(x)| \leq C_2|g(x)| \text{ if } x > k \quad (30)$$

$C$  and  $k$  are called **witnesses**. In particular

#### Big-Theta Definition

$f(x)$  is  $\Theta(g(x))$  if  $f(x)$  is  $\mathcal{O}(g(x))$  and  $f(x)$  is  $\Omega(g(x))$

#### Implication of Theta Notation

When  $f(x)$  is  $\Theta(g(x))$ , then  $g(x)$  is  $\Theta(f(x))$ .

#### Equivalence of Theta Notation

$f(x)$  is  $\Theta(g(x)) \Leftrightarrow f(x)$  is  $\mathcal{O}(g(x))$  and  $g(x)$  is  $\mathcal{O}(f(x))$

### 3.3.10 Big-Theta Estimation for Polynomials

#### Complexity of Polynomials

Let  $p(x) = a_n x^n + a_{n-1} x^{n-1} + \dots + a_1 x + a_0$  be a polynomial with  $a_i \in \mathbb{R}$  and  $a_n \neq 0$ . Then  $p(x)$  is  $\Theta(x^n)$

### 3.3.11 Little-o Notation

The little-o notation describes the growth of functions too but in a different way. We say that  $f(x)$  is  $o(g(x))$  if  $g(x)$  growth faster as  $f(x)$ . In particular,  $f(x)$  is  $o(g(x))$  if

$$\lim_{x \rightarrow \infty} \frac{f(x)}{g(x)} = 0 \quad (31)$$

## 3.4 Complexity of Algorithms

### 3.4.1 Definition

In computer science, we want to know the behaviour of an algorithm for large values when we want to solve a problem. We want to describe the algorithm

in terms of the time to solve a problem depending of the input size. The case that we are interested is when the input size is very big and we want to know if algorithms with this size of inputs are efficient. In computer science, we talk about **computational complexity**. This computational complexity can be separated in 2 parts: **time complexity**, how much an algorithm take time to solve a problem, and **space complexity**, the resources that the algorithm must have to solve a problem. We will focus on the first one.

### 3.4.2 Time Complexity

The time complexity corresponds to the number of operations performed, and we will use the big-O and big- $\Theta$  notations to describe it. For this, we assume that all operations take the same time (addition, multiplication, ...). In the problem of complexity, we will focus on the **Worst-case time** that is, for a given problem with an algorithm to solve it, we will take the upper bound on the number of operations.

#### 3.4.3 Example of Worst-case complexity (1)

if we take the pseudo-code for the max of a list of numbers:

```

procedure max(a_1,...,a_n : integers)
    tmp_max := a_1
    for(i := 2 to n)
        if(tmp_max < a_i) then tmp_max := a_i

    return tmp_max

```

We see that the  $max < a_i$  is made  $n - 1$  times. Each time  $i$  is incremented, we have a comparison to see if  $i \leq n$ , so  $n - 1$ . One last comparison is made to see if  $i > n$ . So in total, we have  $2(n - 1) + 1 = 2n - 1$  made. We can conclude that the complexity of this algorithm is  $\Theta(n)$ .

#### 3.4.4 Example of Worst-case complexity (2)

if we take the pseudo-code for the insertion sort algorithm:

```

procedure insertion_sort(a_1,...,a_n : real numbers with n >= 2)
    for(j := 2 to n)
        i := 1
        while(a_j > a_i and i < j)
            i := i + 1
        m := a_j
        for(k := 0 to j - i - 1)
            a_{j-k} := a_{j-k-1}
        a_i := m

```

We see first that we have  $n - 1$  passes for the first loop from  $j = 2$  to  $n$ . In each pass, the while and the for loop inside are executed at most  $j$  times. Since  $2(2 + 3 + \dots + n) = n(n + 1) - 2$ , the complexity is  $\Theta(n^2)$ .

### 3.4.5 Summary of Complexity for some Algorithms

<i>Algorithm</i>	<i>Time Complexity</i>	<i>Algorithm</i>	<i>Time Complexity</i>
<i>Linear Search</i>	$\Theta(n)$	<i>Binary Search</i>	$\Theta(\log(n))$
<i>Bubble Sort</i>	$\Theta(n^2)$	<i>Insertion Sort</i>	$\Theta(n^2)$

### 3.4.6 Hierarchy of Functions

Non-exhaustive list of complexity of functions

1. Constant:  $\mathcal{O}(1)$
2. Logarithm:  $\mathcal{O}(\log(x))$
3. Poly-logarithm:  $\mathcal{O}(\log(x)^n)$
4. Linear:  $\mathcal{O}(x)$
5. Linear-arithmetic:  $\mathcal{O}(x \log(x))$
6. Polynomial:  $\mathcal{O}(x^n)$
7. Exponential:  $\mathcal{O}(n^x)$

## 4 Induction and Recursion

### 4.1 Mathematical Induction

#### 4.1.1 Definition

The principle of mathematical induction is that we want to prove that something is true for a value  $k+1$  assuming a certain base and that the proposition  $P(n)$  is true for  $P(k)$ . So we have a **basic step** (test the proposition for a small value), the **inductive hypothesis** (we assume that the proposition is true for a  $k$ ) and the **inductive step** (prove that the proposition is true for the value  $k+1$ ).

#### 4.1.2 Induction as Inference Rule

The mathematical induction can be described as the rule of inference

$$(P(1) \wedge \forall k(P(k) \rightarrow P(k+1))) \rightarrow \forall n P(n) \quad (32)$$

#### 4.1.3 Why Induction is Valid ?

The mathematical induction is valid by the well-ordering property (an axiom for the set of positive integers) that says: every non-empty subset of  $\mathbb{N}$  contains a least element. Mathematical induction is equivalent to this.

#### 4.1.4 Example of Induction

We want to prove this following sum

$$P(n) : \sum_{i=1}^n i = \frac{n(n+1)}{2}$$

First the basic step:  $P(1) = \sum_{i=1}^1 i = 1 = \frac{1(2)}{2}$  so it's good. Let assume that the formula is true for  $P(k)$  so that  $\sum_{i=1}^k i = \frac{k(k+1)}{2}$ . Let's start the inductive step:

$$\sum_{i=1}^{k+1} i = \sum_{i=1}^k i + (k+1) = \frac{k(k+1)}{2} + (k+1) = \frac{k(k+1)+2(k+1)}{2} = \frac{(k+1)(k+2)}{2} = P(k+1)$$

#### 4.1.5 Example of Induction with Sets

The theorem is that for a finite set  $S$ , the number of subset of  $S$  is  $2^n$  where  $n$  is it's cardinality. The basic step is for example  $P(0)$  and it's true because the empty set  $\emptyset$  does not have elements and so the unique subset is itself so  $2^0 = 1$ . Now let's note the inductive hypothesis, so that every set has  $2^k$  subsets with  $k$  as cardinality. Now let's move on the inductive step. Let  $T$  be a set with  $k+1$  elements. Let  $S$  be the set as  $T = S \cup a$  for an element  $a \in T$  and so  $S = T - a$ . Hence,  $|S| = k$  (because  $|T| = k+1$ ). Now for each subset  $X$  of  $S$ , we can have two subset of  $T$ :  $X$  and  $X \cup a$ . By inductive step,  $S$  has  $2^k$  subsets. And so since  $T$  has 2 subsets for each subsets of  $S$ , the number of subsets of  $T$  is  $2 * 2^k = 2^{k+1}$ .

#### 4.1.6 Strong Induction

To prove that a proposition  $P(n)$  is true for all  $n$ , we say that

$$\forall k ([P(1) \wedge \dots \wedge P(k)] \rightarrow P(k+1)) \quad (33)$$

We say that the strong induction is the complete mathematical induction.

### 4.2 Recursive Definitions and Structural Induction

#### 4.2.1 Recursive Defined Function

A recursive of a function  $f$ , with non-negative integers as domain, consists of 2 steps

1. Basic step: specify the value of the function at zero.
2. Recursive step: Give a rule for finding the next value from it's last values.  
For example, the function  $f(0) = 3$  and  $f(n+1) = 2f(n) + 1$ .

We can write the basic recursive defined function for sums

$$\sum_{k=0}^{n+1} a_k = \left( \sum_{k=0}^n a_k \right) + a_{n+1} \quad (34)$$

#### 4.2.2 Recursively Defined Sets and Structures

Same as the recursive defined functions, we can say that there is 2 steps:

1. Basic step specifies an initial collection of elements
2. Recursive step gives the rules for forming new elements in the set from those already in the set.

The basic example is the formation of the set  $\mathbb{N}$ .

- Basic step:  $0 \in \mathbb{N}$
- Recursive step: If  $n \in \mathbb{N}$  then  $n+1 \in \mathbb{N}$ .

### 4.3 Recursive Algorithms

#### 4.3.1 Definition

An algorithm is called **recursive** if it solves a problem by reducing it to an instance of the same problem with smaller input. The basic example of this is the factorial algorithm

```
factorial(n):=
  if(n = 0)
    then return 1
  else
    then return n * factorial(n - 1)
```



```

power(a, 6) =
a * power(a, 5) =
a * (a * power(a, 4)) =
a * (a * (a * power(a, 3))) =
a * (a * (a * (a * power(a, 2)))) =
a * (a * (a * (a * (a * power(a, 1)))))) =
a * (a * (a * (a * (a * (a * power(a, 0))))))) =
a * (a * (a * (a * (a * (a * 1)))))) =
a * (a * (a * (a * (a * a)))) =
a * (a * (a * (a * a^2))) =
a * (a * (a * a^3)) =
a * (a * a^4) =
a * a^5 =
a^6

```

(a) Simple Algorithm

```

fast_power(a, 6) =
a^0 * fast_power(a, 3)^2 =
a^0 * (a^1 * fast_power(a, 1)^2) =
a^0 * (a^1 * (a^1 * fast_power(a, 0)^2)) =
a^0 * (a^1 * (a^1 * (a^0)^2)) =
a^0 * (a^1 * (a^1 * 1)) =
a^0 * (a^1 * a^2) =
a^0 * (a^3)^2 =
a^6

```

(b) Better Algorithm

Figure 8: Comparison between 2 recursive algorithms for computing powers of a number

### 4.3.2 Recursive Algorithms Better Than Others

Let's write a recursively algorithm for computing powers of a number.

```

power(a,n):=
  if(n <= 1)
    then return 1
  else
    then return a * power(a,n-1)

```

But this algorithm is not really good. Let's write a better algorithm:

```

nice_power(a,n):=
  if(n <= 1)
    then return 1
  else
    then return (a power n&1) *
    nice_power(a, floor of n/2) square

```

### 4.3.3 Difference between Recursion and Induction

Recursion and Induction are not the same things when we talk about resolving a problem. The first reduces a problem of any size to the smallest one possible. The second extends this ability to problems of any size. In consequence, a recursively defined function can be described in two ways: recursively or iteratively.

```

iterative_fibonacci(n):=
  if(n=0) then return 1
  else
    previous := 0
    current := 1
    for(i=0 to n-1)
      next := previous + current
      previous := current
      current := next
    return current

```

```

recursive_fibonacci(n):=
  if(n <= 1) then return 1
  else return recursive_fibonacci(n-1) +
    recursive_fibonacci(n-2)

```

## 5 Number Theory and Group Theory

### 5.1 Division

#### 5.1.1 Division

Let  $a, b \in \mathbb{Z}$  and  $a \neq 0$ . We say that a number  $a$  divides  $b$  if there is an integer  $c$  such that  $b = ac$ . We use the notation  $a|b$  to say that  $a$  divides  $b$ .

#### 5.1.2 Properties of Division

We can list some properties of division

- $a|c$  and  $a|b$  then  $a|(b + c)$
- $a|c$  then  $a|bc$ ,  $\forall b \in \mathbb{Z}$
- $a|c$  and  $c|b$  then  $a|b$  (transitivity)

Proof for the first:

Suppose that  $a|b$  and  $a|c$ . So there exists integers  $s$  and  $t$  such that  $b = as$  and  $c = at$ . Hence  $b + c = as + at = a(s + t)$ . Hence  $a|(b + c)$

Let  $a, b \in \mathbb{N}_+$ . if  $a|c$  and  $b|c$ , it **not** implies that  $ab|c$ . To have that we must have  $\gcd(a, b) = 1$ .

### 5.1.3 Division Algorithm

#### Division Algorithm

If  $a$  is an integer and  $d$  a positive integer, then there are unique integers  $q$  and  $r$  with  $0 \leq r < d$  such that  $a = dq + r$

## 5.2 Modular Arithmetic

### 5.2.1 Congruence Relation

If  $a, b \in \mathbb{Z}$  and  $m > 0$ , then we say that  $a$  is congruent to  $b$  modulo  $m$  if  $m|a - b$  and we write  $a \equiv b \pmod{m}$ . For example,  $17 \equiv 5 \pmod{6}$  because  $6|17 - 5$ .

#### Congruence Equivalence

$$a \equiv b \pmod{m} \Leftrightarrow a \pmod{m} = b \pmod{m}$$

### 5.2.2 Congruences of Sums and Products

#### Congruence of Sums and Products

Let  $m > 0$  with  $a \equiv b \pmod{m}$  and  $c \equiv d \pmod{m}$  then

- $a + c \equiv (b + d) \pmod{m}$
- $ac \equiv (bd) \pmod{m}$

Dividing a congruence by an integer will not always produce a valid congruence!

## 5.3 Integer Representation and Algorithm

### 5.3.1 Representation of Integers

In general, the representation used is called the base 10 representation and it means that every numbers can be written with powers of 10. For example,  $534 = 5 * 10^2 + 3 * 10^1 + 4 * 10^0$ .

### 5.3.2 Base b Representation

Let  $b \in \mathbb{N}$  and  $b > 1$ . Then if  $n$  is a positive integer, it can be written uniquely in the form

$$n = a_k b^k + a_{k-1} b^{k-1} + \dots + a_1 b + a_0 \quad (35)$$

We write then that  $n = (a_k, a_{k-1}, \dots, a_1, a_0)_b$ . We have many bases, but the principals are base 2 (binary), 8 (octal), 10 (decimal) and 16 (hexadecimal).

### 5.3.3 Base $b$ Expansion Algorithm

```
expansion(n,b: positive integer, b > 1)
q := n
k := 0
while(q different of 0)
    a_k := q mod b
    q := q div b (dividend)
    k := k + 1
return (a_k,a_{k-1},...,a_0)
```

The digits in the base  $b$  expansion are the remainders of the division given by  $q \pmod{b}$

### 5.3.4 Conversion into Basis

There are different ways to switch into basis, but some are more efficient than others. Some conversions for the binary expansion  $(11111010111100)_2$  :

1. In Octal we group the bits by 3 and add initial 0 if it's necessary. here we have  $(011\ 111\ 010\ 111\ 100)$  and so the groups represent a number in base 8:  $(37274)_8$ .
2. In Hexadecimal we group the bits by 4 and add initial 0 if it's necessary. here we have  $(0011\ 1110\ 1011\ 1100)$  and so the groups represent a number in base 16:  $(3EBC)_{16}$ .

## 5.4 Arithmetic with Base 2

### 5.4.1 Addition of Integers

The algorithm below is  $\mathcal{O}(n)$  and it computes the addition of 2 integers in base 2:  $a = (a_{n-1}, \dots, a_0)_2$  and  $b = (b_{n-1}, \dots, b_0)_2$ .

```
addition(a,b: positive integers)
c := 0
for(j := 0 to n - 1)
    d := (a_j + b_j + c)/2 and then the floor
    s_j := a_j + b_j + c - 2d
    c := d
s_n := c
return (s_0,...,s_{n})
```

The binary expansion of the sum is now  $(s_n, \dots, s_0)_2$

### 5.4.2 Multiplication of Integers

The algorithm below is  $\mathcal{O}(n^2)$  and it computes the multiplication of 2 integers in base 2:  $a = (a_{n-1}, \dots, a_0)_2$  and  $b = (b_{n-1}, \dots, b_0)_2$ .

```
multiplication(a,b: positive integers)
for(j := 0 to n - 1)
    if b_j = 1 then c_j := a with j zero appended
    else c_j := 0
{c_0, ..., c_{n-1} are the partial products}
p := 0
for(j := 0 to n - 1)
    p := p + c_j
return p
```

The binary expansion of the sum is now  $(s_n, \dots, s_0)_2$

We see that  $a * b = a * (b_k 2^k + \dots + b_1 2 + b_0) = ab_k 2^k + \dots + ab_1 2 + ab_0$ .  
In conclusion, multiplying a binary number by  $2^j$  corresponds to add  $j$  zeros at the end.

## 5.5 Prime Numbers

### 5.5.1 Definition

A number  $a$  is said prime if and only if the only numbers that divides  $a$  are 1 and itself. A number not prime is called composite.

### 5.5.2 Fundamental Theorem of Arithmetic

Decomposition of Numbers in primes product

If  $n \in \mathbb{Z}_+ > 1$ , then  $n$  can be written as product of prime numbers.

Proof:

Proof by strong induction. Let  $P(n)$  be the proposition that  $n$  can be written as product of primes. The basic step is true because  $P(2)$  is true. The inductive hypothesis is  $P(j)$  is true for all integers  $2 \leq j \leq k$ . To show that  $P(k+1)$  must be true under this assumption, there are 2 cases.

1. If  $k+1$  is prime, then  $P(k+1)$  is true
2. Otherwise,  $k+1$  is composite and can be written as product of 2 positive integers  $a$  and  $b$  with  $2 \leq a \leq b < k+1$ . By the inductive hypothesis,  $a$  and  $b$  can be written as product of primes and therefore  $k+1$  can be also written as the product of primes.

Hence, it has been shown that every integer greater than 1 can be written as product of primes.

### 5.5.3 Trial Division

#### Trial Division

If  $n$  is a composite number, then  $n$  has a prime divisor less than or equal to  $\sqrt{n}$

### 5.5.4 Infinity of Primes

#### Infinity of Primes

There are infinitely many primes (Euclid)

Proof:

Assume finitely primes  $p_1, \dots, p_n$ . Let  $q = p_1 * p_2 * \dots * p_n + 1$ . Either  $q$  is prime or by the fundamental theorem of arithmetic it is a product of primes.

- None of the primes  $p_j$  divides  $q$  since if  $p_j | q$  then  $p_j$  divides  $q - p_1 * \dots * p_n = 1$ .
- Hence, there is a prime not in the list  $p_1, \dots, p_n$ .
- It is either  $q$ , or if  $q$  is composite, it is a prime factor of  $q$ .
- This contradict the assumption that  $p_1, \dots, p_n$  are all primes.

Consequently, there are infinitely many primes.

### 5.5.5 Distribution of Primes

We note  $\pi(x)$  to denote the **number of primes not exceeding  $x$** . The function describes an asymptotic distribution of the primes and shows that primes become less common as they become larger. We can note that

$$\pi(x) \approx \frac{x}{\ln(x)} \quad (36)$$

#### Prime Number Theorem

$$\lim_{x \rightarrow \infty} \frac{\pi(x)}{\frac{x}{\ln(x)}} = 1 \quad (37)$$

### 5.5.6 Theorem (Prime Numbers)

Every integer  $> 1$  can be written as a sum of product of prime numbers. And finding this representation for large number is hard. Conclusion

$$P_1, P_2 \rightarrow P_1 * P_2 \text{ is a one way function} \quad (38)$$

### 5.5.7 Co-prime

If  $\gcd(a, b) = 1$  then  $a$  and  $b$  are co-prime.

### 5.5.8 Theorem (Co-prime)

$a, b \in \mathbb{N}_+$ . If  $a|c$  and  $b|c$  and the  $\gcd(a, b) = 1$ , then  $ab|c$ .

### 5.5.9 Goldbach's Conjecture

The goldbach's conjecture is one of the oldest conjecture in number theory, introduced in 1742 by the mathematician Christian Goldbach. The conjecture said that:

Every even integer  $n$  greater than 2 is the sum of 2 primes.

It has been tested for very large integers but there is still no rigorous proof.

## 5.6 GCD - LCM

### 5.6.1 Greatest Common Divisor (GCD)

Let  $a$  and  $b$  be integers, non both 0. The largest integer  $d$  such that  $d|a$  and  $d|b$  is called the greatest common divisor and it's denoted with

$$d = \gcd(a, b) \quad (39)$$

### 5.6.2 Relatively Prime

We say that  $a \in \mathbb{Z}$  and  $b \in \mathbb{Z}$  are relatively prime if and only if

$$\gcd(a, b) = 1 \quad (40)$$

### 5.6.3 Compute the GCD

To compute the gcd of 2 integers, there are different ways. The first uses the prime factorisation of the 2 integers. The second is applying a great relation between 2 gcd. The first is interesting because it's simple: We write  $a = p_1^{\alpha_1} p_2^{\alpha_2} \dots p_n^{\alpha_n}$  and  $b = p_1^{\beta_1} p_2^{\beta_2} \dots p_n^{\beta_n}$ , with all  $p_i$  primes. Then the greatest common divisor of  $a$  and  $b$  is

$$\gcd(a, b) = p_1^{\min(\alpha_1, \beta_1)} \dots p_n^{\min(\alpha_n, \beta_n)} \quad (41)$$

But this method can be really awful with large numbers because we have to know all primes less or equal than  $a$  and  $b$ . The second way is to use the fact that for an integer  $a = bq + r$

$$\gcd(a, b) = \gcd(b, r) \quad (42)$$

The proof is really simple

- Suppose that  $d$  divides both  $a$  and  $b$ . Then  $d$  divides also  $a - bq = r$ . Hence, any common divisor of  $a$  and  $b$  must also be a common divisor of  $b$  and  $r$ .
- Suppose that  $d$  divides both  $b$  and  $r$ . Then  $d$  divides also  $bq + r = a$ . Hence, any common divisor of  $b$  and  $r$  must also be a common divisor of  $a$  and  $b$ .
- Hence  $\gcd(a, b) = \gcd(b, r)$

This algorithm is called **the Euclidean Algorithm**

### 5.6.4 Complexity of the Euclidean Algorithm

#### Lamé's Theorem

Let  $a$  and  $b$  be positive integers with  $a \geq b$ . Then the number of divisions used by the Euclidean Algorithm is less or equal to five times the number of decimal digits in  $b$ . Therefore, it has complexity  $\mathcal{O}(\log(n))$ .

### 5.6.5 Least Common Multiple (LCM)

Let  $a$  and  $b$  be integers. The least common multiple of  $a$  and  $b$  is the smallest positive integer  $d$  that is both divisible by  $a$  and  $b$ . It's denoted

$$d = \text{lcm}(a, b) \quad (43)$$



### 5.6.6 Compute the LCM

The algorithm is really near of the algorithm of the GCD. We write  $a = p_1^{\alpha_1} p_2^{\alpha_2} \dots p_n^{\alpha_n}$  and  $b = p_1^{\beta_1} p_2^{\beta_2} \dots p_n^{\beta_n}$  as prime factors. Then the least common multiple of  $a$  and  $b$  is

$$\gcd(a, b) = p_1^{\max(\alpha_1, \beta_1)} \dots p_n^{\max(\alpha_n, \beta_n)} \quad (44)$$

### 5.6.7 Multiplying GCD and LCM

let  $a$  and  $b$  be integers. Then

$$ab = \gcd(a, b) * \text{lcm}(a, b) \quad (45)$$

## 5.7 Congruence Classes

### 5.7.1 Definition of Modulus

Let  $m > 1$  be an integer called modulus. Let  $a \in \mathbb{Z}$ ,  $[a]_m = [i \in \mathbb{Z} : i \equiv a \pmod{m}]$  is the congruence class of  $a \pmod{m}$ . Example:

- $[1]_2 = [\dots, -1, 1, 3, 5, \dots]$  the set of odd integers.
- $[a]_m = [b]_m$  iff  $a \equiv b \pmod{m}$ .

### 5.7.2 Set of All Congruence Classes

The set of all congruence classes is denoted by  $\mathbb{Z}/m\mathbb{Z}$ . Every element of this class has a unique representation in reduced form: If  $r = a \pmod{m}$  then  $[r]_m = [a]_m$  where  $[r]_m$  is the reduced form.

- $[22]_7 = [8]_7$  because  $7 \mid 22 - 8$ .

### 5.7.3 Properties of congruence classes

- $[a]_m + [b]_m = [a + b]_m$
- $[a]_m * [b]_m = [a * b]_m$
- $[a]_m + [0]_m = [0]_m$
- $[a]_m + [1]_m = [a]_m$
- $([a]_m)^k = [a]_m * \dots k \text{ times}$
- $([a]_m)^{-k} = ([a]_m)^{-1} * \dots k \text{ times}$
- $([a]_m)^0 = [1]_m$

#### 5.7.4 Inverse and Solution Equivalence

In  $\mathbb{Z}/n\mathbb{Z}$ , the following statements are equivalent

- $a$  has an inverse.
- $\forall b \in \mathbb{Z}/n\mathbb{Z}, ax = b$  has a unique solution.
- $\exists b \in \mathbb{Z}/n\mathbb{Z}$  such that  $ax = b$  has a unique solution.

#### 5.7.5 Theorem of Multiplicative Inverse

##### Existence of Multiplicative Inverse

Let  $[a]_m \in \mathbb{Z}/m\mathbb{Z}^*$ , then  $[a]_m$  has a multiplicative inverse if and only if  $\gcd(a, m) = 1$ .

#### 5.7.6 Multiplicative Inverse and Primes

##### Existence of Multiplicative Inverse

If  $p$  is a prime number, all non-zero elements of  $\mathbb{Z}/p\mathbb{Z}$  have a multiplicative inverse.

The proof is really simple

For every  $a \in \{1, 2, 3, \dots, p-1\}$ ,  $\gcd(a, p) = 1$  and then  $\gcd(0, p) = p$ .

#### 5.7.7 Bézout Identity

##### Bézout Identity

Let  $a, b \in \mathbb{Z}$  not both 0. There exists integers  $u, v$  such that  $\gcd(a, b) = au + bv$ . So  $\gcd(a, b) = 1$  if and only if  $\exists$  integers such that  $au + bv = 1$ .

The proof is really important because the method that we use in the RSA chapter to compute some important numbers follows the proof.

Suppose that  $a \geq b$

- $\gcd(a, b) = \gcd(b, r)$ , where  $a = bq + r$
- Suppose we found  $\tilde{u}$  and  $\tilde{v}$  such that  $\gcd(b, r) = b\tilde{u} + r\tilde{v}$ .
- Now let's use  $r = a - bq$  and we rewrite  $\gcd(a, b) = \gcd(b, r) = b\tilde{u} + r\tilde{v} = b\tilde{u} + (a - bq)\tilde{v} = a\tilde{v} + b(\tilde{u} - q\tilde{v}) = au + bv$  comparing terms  $u = \tilde{v}$  and  $\tilde{u} = (\tilde{u} - q\tilde{v})$ .
- The final step is  $\gcd(a, 0) = a \Rightarrow \tilde{u} = 1$  and  $\tilde{v} = 0$ . So  $\tilde{v}$  is not unique.
- Via successive application of this, we find that  $\gcd(a, b) = au + bv$

$\gcd(a, b)$	$a = bq + r$	$u = \tilde{v}$	$v = (\tilde{u} - q\tilde{v})$
$\gcd(122, 22)$	$122 = 22 \times 5 + 12$	2	$-1 - 5 \times 2 = -11$
$\gcd(22, 12)$	$22 = 12 \times 1 + 10$	-1	2
$\gcd(12, 10)$	$12 = 10 \times 1 + 2$	1	-1
$\gcd(10, 2)$	$10 = 2 \times 5 + 0$	0	1
$\gcd(2, 0) = 2$		1	0

$\gcd(122, 22) = 122 \times 2 + 22 \times (-11)$

Figure 9: Methodology to search the numbers such that the  $\gcd(a, b) = au + bv$  by the proof of Bezout Identity

## 5.8 Commutative Groups

### 5.8.1 Definition

A commutative groups (Abelian group) is a set  $G$  with a binary operation  $*$  that satisfies axioms: closure ( $ab \in G$ ), associativity, commutativity, identity. For example:  $(\mathbb{R}, +)$  but not  $(\mathbb{R}, *)$  because 0 does not have an inverse,  $(\mathbb{Z}/n\mathbb{Z} - [0]_n, *)$  when  $n$  is a prime number.

### 5.8.2 Euler's phi function

Euler phi's function is denoted by  $\phi(n)$  and it's the number of positive integers in  $[1, 2, \dots, n]$  that are relatively prime to  $n$ . For example:  $\phi(3) = 2$  because 1 is prime with 3, 2 too but not 3.

- $\phi(n)$  is the cardinality of  $\mathbb{Z}/n\mathbb{Z}^*$ .
- If  $p$  is prime, then  $\phi(p) = p - 1$ .
- In  $[1, 2, \dots, p^k]$ ,  $p$  is prime and  $k \in \mathbb{Z}$ , the elements that are not relatively prime to  $p^k$  are:  $p, 2p, \dots, (p^{k-1})p$ . So there are  $p^{k-1}$  elements so  $\phi(p^k) = p^k - p^{k-1}$ .
- Similarly, for  $[1, \dots, pq]$ , then  $\phi(pq) = pq - q - (p - 1) = \phi(q)\phi(p)$ .

### 5.8.3 Cartesian Product with Groups

Let  $(G_1, *_1)$  be a set with a binary operation (not necessary commutative group), and similarly for  $(G_2, *_2)$ . We define  $(G, *) = (G_1, *_1) \times (G_2, *_2)$  where the binary operation  $*$  is defined by  $(a_1, a_2) * (b_1, b_2) = ((a_1 *_1 b_1), (a_2 *_2 b_2)) \in G$ .

For example:  $(\mathbb{Z}/2\mathbb{Z}, +) \times (\mathbb{Z}/3\mathbb{Z}^*, *)$ . With this we have  $\mathbb{Z}/2\mathbb{Z} = [0, 1]$  and  $\mathbb{Z}/3\mathbb{Z}^* = [1, 2]$  so we compute the array for each multiplication modulo 3.

Note that if the 2 sets are commutative groups, then the cartesian product is also a commutative group.

## 5.9 Isomorphisms

### 5.9.1 Definition

#### Definition of Isomorphism

Let  $(G, \triangle)$  and  $(H, \oplus)$  be sets, each endowed with a binary operation. An isomorphism from  $(G, \triangle)$  to  $(H, \oplus)$  is a bijection  $\phi : G \rightarrow H$  such that

$$\phi(a \triangle b) = \phi(a) \oplus \phi(b) \quad (46)$$

holds for all  $a, b \in G$ . We say that the sets are isomorphic.

### 5.9.2 Implication of Isomorphism

If 2 sets are isomorphic, then:

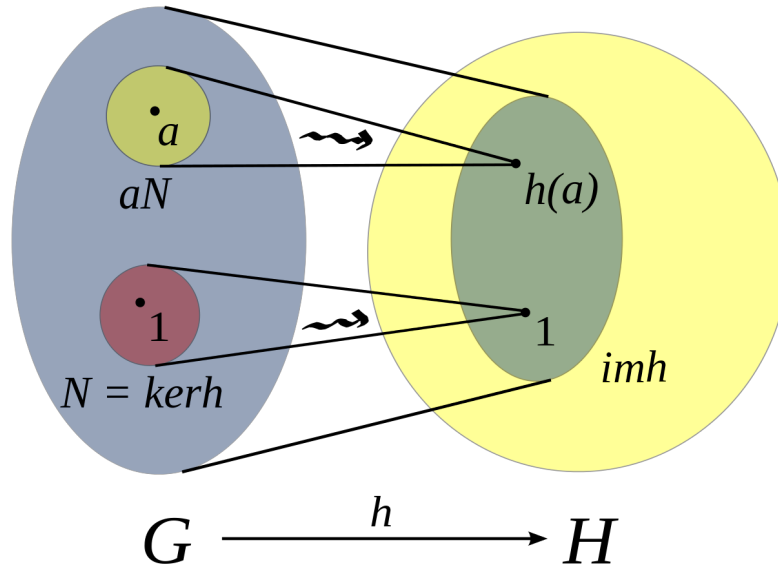


Figure 10: Representation of an Isomorphism from  $G$  to  $H$ . The smaller oval is the image of  $h$ ,  $N$  is the kernel of  $H$  and  $aN$  is a coset of  $N$

- if  $(H_1, *_1)$  is a commutative group, then  $(H_2, *_2)$  is also a commutative group.
- If  $e$  is the identity element of  $(H_1, *_1)$ , then  $\phi(e)$  is the identity element of  $(H_2, *_2)$ .
- If  $a, b$  are inverse of one-another in  $(H_1, *_1)$ , then  $\phi(a)$  and  $\phi(b)$  are inverse of one-another in  $(H_2, *_2)$ .

This is the proof for the first point:

Each element of  $H$  is written  $\phi(x)$  for some  $x \in G$ .

- Closure:  $\phi(a) \oplus \phi(b) = \phi(a \triangle b) \in H$
- Associativity: No matter in which order we perform the operations on the left and side,  $\phi(a) \oplus \phi(b) \oplus \phi(c) = \phi(a \triangle b \triangle c) \in H$
- Identity element:  $\phi(e) \oplus \phi(a) = \phi(e \triangle a) = \phi(a)$  proving that  $\phi(e)$  the identity element of  $(H, \oplus)$
- Inverse element:  $\phi(a) \oplus \phi(a^{-1}) = \phi(a \triangle a^{-1}) = \phi(e)$ , showing that the inverse of  $\phi(a)$  is  $\phi(a^{-1})$
- Commutativity:  $\phi(a) \oplus \phi(b) = \phi(a \triangle b) = \phi(b \triangle a) = \phi(b) \oplus \phi(a)$

### 5.9.3 Definition (Order)

Let  $(G, *)$  be a finite commutative group and let  $e \in G$  be the identity element. For any element  $a \in G \exists k \in \mathbb{N}_+$  such that  $a * a * a \dots * a = e$ ,  $k$  times. And when we talk about the order of a group, it's his cardinality.

#### Order of an element

The smallest  $k \in \mathbb{N}_+$  such that  $a^k = e$  is called the order of  $a$ .

For example, in  $(\mathbb{Z}/3\mathbb{Z}, +)$ , the order of the element  $[1]_3$  is 3 because the have to do  $[1]_3 + [1]_3 + [1]_3$  to have the identity element, so here  $[0]_3$ .

The orders are something extremely important in group theory because it gives us a lot of informations about the group itself. For example, the more complicated factorization of the cardinality of  $G$ , the more complicated the structure is. An other example, and it's very useful for the chapter of cryptography, is that we can count the number of numbers with an order  $p$ . By lagrange's Theorem, we know that an order of an element divides the cardinality of a set  $G$ . The number of element with order  $p$  is a multiple of  $\phi(p)$ , the euler totient function.

An other example is the usage of orders to know if an isomorphism can exist between 2 finite groups. Let  $f : G \rightarrow H$  a function between 2 finite groups. If  $f$  is an isomorphism, then the order of  $f(a)$  divides the order of  $a$ . If  $f$  is injective, then the order of  $f(a)$  equal the order of  $a$ . And this can be often used to prove that there are no isomorphism between 2 groups. For example a function between  $\mathbb{Z}/3\mathbb{Z} \rightarrow \mathbb{Z}/5\mathbb{Z}$  is impossible because every number in  $\mathbb{Z}/5\mathbb{Z}$  have order 5 which does not divides the order 1,2 or 3 of  $\mathbb{Z}/3\mathbb{Z}$ .

#### 5.9.4 Lagrange's Theorem

##### Lagrange's Theorem

Let  $(G, *)$  be a finite commutative group of cardinality  $n$ . The orders of each of its elements divides  $n$ .

#### 5.9.5 Isomorphism and Orders

##### Relation with Isomorphism and orders

2 finite commutative groups are isomorphic if and only if they have the same set of orders.

#### 5.9.6 Equivalence with Lagrange Theorem

Let  $a$  be an element of  $(G, *)$  of order  $p$ . Then

$$a^k = e \Leftrightarrow p|k \quad (47)$$

#### 5.9.7 Euler's Theorem

##### Euler's Theorem

Let  $m > 1$  be an integer. For all  $a \in (\mathbb{Z}/m\mathbb{Z}^*, *)$

$$a^{\phi(m)} = [1]_m \quad (48)$$

#### 5.9.8 Corrolary to Euler (Fermat's Theorem)

##### Fermat's Theorem

Let  $p$  be prime. For all  $a \in (\mathbb{Z}/p\mathbb{Z}, *)$

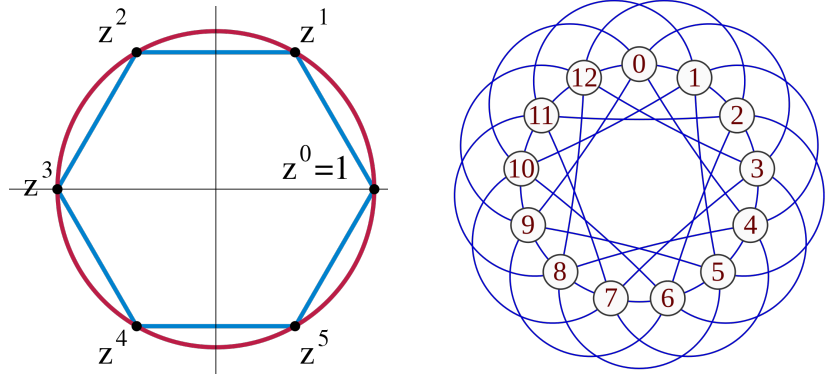
$$a^p \equiv a \pmod{p} \quad (49)$$

#### 5.9.9 The Chinese Remainder Theorem (CRT)

If  $m_1$  and  $m_2$  are relatively prime then the map

$$\phi : \mathbb{Z}/m_1m_2\mathbb{Z} \rightarrow \mathbb{Z}/m_1\mathbb{Z} \times \mathbb{Z}/m_2\mathbb{Z} \quad (50)$$

is bijective and is an isomorphism with respect  $+$  and  $*$ .



(a) The six 6th complex root of unity form a cyclic group under multiplication. Here  $z$  is the generator but not  $z$  square because it's powers fail to produce odd powers of  $z$

(b) The Paley graph of order 13, a circulant graph formed by the Cayley graph of  $\mathbb{Z}/13\mathbb{Z}$  with generators 1,3,4

Figure 11

#### 5.9.10 Theorem (Combine Fermat and CRT)

Let  $p, q$  be distinct prime numbers and let  $k$  be a multiple of both  $p - 1$  and  $q - 1$ . Then  $\forall a, e \in \mathbb{Z}$

$$[a]_{pq}^{ke+1} = [a]_{pq} \quad (51)$$

#### 5.9.11 Cyclic Groups

A cyclic group of order  $n$  is a group  $(G, *)$  of cardinality  $n$  such that, for some  $g \in G$ , called generator,  $G = [g, g^2, g^3, \dots, g^n = e]$ . A cyclic group is always commutative and  $\mathbb{Z}/n\mathbb{Z}$  is a cyclic group.

There are many fact on them. If  $(G, *)$  and  $(H, *)$  are cyclic groups of the same orders, they are isomorphic, every finite cyclic group is isomorphic to the additive group  $\mathbb{Z}$  and for also every cyclic group of cardinality  $n$  is isomorphic to the the group  $\mathbb{Z}/n\mathbb{Z}$ .

Also, with cyclic groups, we have a simple group because if it's cardinality is a prime number, then it cannot be divided in subgroups.

#### 5.9.12 Discrete Exponentiation/Logarithms

Let  $(G, *)$  a cyclic group of order  $n$  and a generator  $b$ . The discrete exponentiation to the base  $b$  is:

$$f : \mathbb{Z}/n\mathbb{Z} \rightarrow G$$

$$[i]_n \rightarrow b^i$$



## 5.10 Fields

A field is a triple  $(K, +, *)$  where  $K$  is a set and  $+, *$  are binary operations that satisfy the following axioms

- $(K, +)$  is a commutative group. We denote the inverse of element  $b \in K$   $-b$  and the identity element is 0.
- $(K - \{0\}, *)$  is a commutative group. We denote the inverse of element  $b \in K - \{0\}$   $b^{-1}$  and the identity element is 1.
- Associativity, Distributivity and Commutativity

If  $K$  is finite, then we have a finite field. Some examples are  $(\mathbb{R}, +, *)$ ,  $(\mathbb{C}, +, *)$ .

The fields  $(\mathbb{Z}/p\mathbb{Z}, +, *)$  are finite fields if and only if  $p$  is prime.

### 5.10.1 Lemma of Prime Numbers (Characteristic)

For every finite field, the addition order of 1 must be a prime number. This number is called the characteristic of the field. The addition order is like how many time do you have to sum 1 to have the 0 (for  $(\mathbb{Z}/2\mathbb{Z}, +, *)$ , the **characteristic** is 2 because  $1 + 1 = 0$  here).

### 5.10.2 Definition Isomorphism in Fields

Isomorphism with fields

An isomorphism between 2 fields  $\mathcal{F} = (F, +, *)$  and  $\mathcal{K} = (K, \otimes, \odot)$  is a bijection  $\phi : F \rightarrow K$  such that  $\forall a, b \in F$

$$\phi(a + b) = \phi(a) \otimes \phi(b) \quad (52)$$


And

$$\phi(a * b) = \phi(a) \odot \phi(b) \quad (53)$$

### 5.10.3 Interesting Facts

- The cardinality of a finite field is an integer power of it's characteristic (hence, all finite fields have cardinality  $p^m$  for some prime  $p$  and  $m \in \mathbb{Z}$ ).
- All finite fields of some cardinality are isomorphic
- $\forall$  prime number  $p$  and  $\forall m \in \mathbb{Z}$ , there is a finite of cardinality  $p^m$ .

The finite field with cardinality  $p^m$  is denoted  $\mathcal{F}_{p^m}$ .



+	0	1	a	b
0	0	1	a	b
1	1	0	b	a
a	a	b	0	1
b	b	a	1	0

Figure 12: Example of operations with the field containing the elements 0,1,a and b

## 5.11 Vector Space

### 5.11.1 Definition

An non-empty set  $V$  is a vector space over a finite field  $\mathcal{F}$  if

- There is a binary operation on  $V$  called vector addition denoted by  $+$  such that  $\forall a, b \in V a + b \in V$ .
- There exists a mixed operation called a scalar multiplication  $\forall \alpha \in \mathcal{F}, v \in V \alpha v \in V$ .

Such that

- $(V, +)$  is a commutative group.
- Associativity, Distributivity, Identity element 1, ...

For example,  $V = (\mathcal{F}^n, +, *)$  where  $+$  is element-wise operation in the field. We say that  $\mathcal{F}_7^3$  is a field with 3 for the vector length and 7 for the field size.

### 5.11.2 Subspace

We say that  $S \subset V$  is a subspace if it is closed under vector condition and scalar multiplication.

- For every  $\vec{v} \in S : \alpha \vec{v} \in S \forall \alpha \in \mathbb{Z}$
- For every  $\vec{v}_1, \vec{v}_2 \in S : \vec{v}_1 + \vec{v}_2 \in S \forall \vec{v}_1, \vec{v}_2 \in S$ .

### 5.11.3 Linear Independence

A linear independence of a list of vector  $[\vec{v}_1, \dots, \vec{v}_n]$  is a vector of the form  $\vec{v} = \sum_{i=1}^n \lambda_i \vec{v}_i$ . The span of a list of vector is the set of all linear combinations. The list spans  $S$ .

A vector space  $V$  is finite-dimensional if there is a finite list of vectors in  $V$  that spans  $V$ .

The vectors  $[\vec{v}_1, \dots, \vec{v}_n]$  are said linear independent if

$$\sum_{i=1}^n \lambda_i \vec{v}_i = 0 \Rightarrow \lambda_i = 0 \quad \forall i = 1, \dots, n \quad (54)$$

A basis for a finite-dimensional vector space  $V$  is a list of vectors that are linearly independent and spans  $V$ .

#### 5.11.4 Basics Theorems

- A list  $[\vec{v}_1, \dots, \vec{v}_n]$  is a basis of  $V \iff$  every  $\vec{v} \in V$  can be written uniquely as  $\vec{v} = \sum_{i=1}^n \lambda_i \vec{v}_i$ .
- Every list that spans  $V$  can be reduced as a basis.
- Every finite-dimensional vector space has a basis.
- Any 2 basis for the same vector space  $V$  have the same number of basis vectors called the dimension of  $V$  and written  $\dim(V)$ .

#### 5.11.5 Cardinality of a vector space

Cardinality of vector spaces

Let  $V$  be a  $n$ -dimensional vector space over  $\mathcal{F}$ .  $V$  has cardinality  $(\text{car}(\mathbf{F}))^n$ .

#### 5.11.6 Subspace $S$ of $\mathbf{F}$ power $n$

It can be described in 2 ways

- By a list of vectors that spans  $S$ .
- By a system of linear homogeneous equations.

#### 5.11.7 Fulfilled a system

A vector  $\vec{v} = (v_1, \dots, v_n) \in \mathcal{F}^n$  fulfils a system of linear homogeneous equations with coefficient matrix  $A \in \mathcal{F}$  if it satisfies  $\vec{v}(A)^T = \vec{0}$ . There set of solutions in  $\mathcal{F}^n$  of a set of linear homogeneous equations in  $n$  variables and coefficients in  $\mathcal{F}$  is a subspace  $S$  of  $\mathcal{F}^n$ .

### 5.11.8 Find the Dimension of a Subspace

Let  $V \subseteq \mathcal{F}_7^3$  and  $V = \{(x_1, x_2, x_3) \in \mathcal{F}_7^3 : 3x_1 + 2x_2 + x_3 = 0\}$ . What is its dimension ?

The subspace is spanned by the vector  $(3, 2, 1)$ , and especially we know that  $(3, 2, 1)(x_1, x_2, x_3)^T = 0$ . First we notice that there are 2 free variables. We put so  $x_1 = \alpha$  and  $x_2 = \beta$  and so  $x_3 = -3\alpha - 2\beta$  and so:  $(x_1, x_2, x_3) = (\alpha, \beta, -3\alpha - 2\beta) = (1, 0, 4)\alpha + (0, 1, 5)\beta$  (because we are modulo 7). Clearly  $\vec{v}_1 = (1, 0, 4)$  and  $\vec{v}_2 = (0, 1, 5)$  are independent and are in  $V$ . The dimension of the subspace is 2.

### 5.11.9 Important Facts

- The set of solutions in  $V = \mathcal{F}^n$  of  $m$  linear homogeneous equations in  $n$  variables is a subspace  $S$  of  $V$ .
- Let  $r$  be the dimensionality of a vector space spanned by the coefficient vectors. Then  $\dim(S) = n - r$ .
- In particular, if the  $m$  vectors of coefficients are linearly independent, then  $\dim(S) = n - m$ .
- Conversely, if  $S$  is a subspace of  $V = \mathcal{F}^n$  with  $\dim(S) = k$ , there exists a set of  $n - k$  linear equations with coefficients that form linearly independent vectors in  $V$ , the solutions of which are the vectors in  $S$ .

### 5.11.10 Find the Dimension of a Subspace (Revisited)

Let  $V \subseteq \mathcal{F}_7^3$  and  $V = \{(x_1, x_2, x_3) \in \mathcal{F}_7^3 : 3x_1 + 2x_2 + x_3 = 0\}$ . What is its dimension ?

There's only one vector with coefficients:  $(3, 2, 1)$ , so it spans a vector space of dimension  $r = 1$ . Then we know now that  $\dim(V) = n - r = 3 - 1 = 2$ . By this, we fix a canonical basis for the vectors  $\vec{v}_1 = (1, 0, v_{11})$  and  $\vec{v}_2 = (0, 1, v_{21})$ . We replace  $\vec{v}_1$  in the equation above and we find  $v_{11} = -3 = 4$  and same for  $v_{21}$ . We find then  $(1, 0, 4)$  and  $(0, 1, 5)$ .

### 5.11.11 Cardinality and Dimension

Cardinality of a vector space

An  $n$ -dimensional vector space  $V$  over a finite field  $\mathcal{F}$  has cardinality

$$|V| = |\mathcal{F}|^n \quad (55)$$

The proof is straightforward

Let  $(\vec{v}_1, \dots, \vec{v}_n)$  be a basis of  $V$ . For every  $\vec{v} \in V$  there is a unique  $n$ -tuple  $(\lambda_1, \dots, \lambda_n) \in \mathcal{F}^n$  such that  $\vec{v} = \sum_i \lambda_i \vec{v}_i$ . Hence the mapping

$$\mathcal{F}^n \rightarrow V \text{ with } (\lambda_1, \dots, \lambda_n) \mapsto \sum_i \lambda_i \vec{v}_i$$

is a bijection. By the pigeonhole principle,  $|V| = |\mathcal{F}^n| = |\mathcal{F}|^n$

## 6 Counting

### 6.1 Basic Counting Principles

#### 6.1.1 Product Rule Principle

Let  $A$  and  $B$  2 tasks with  $n_1$  and  $n_2$  ways to do  $A$  and  $B$  respectively. Then the number of ways to do the 2 tasks is

$$n = n_1 n_2 \quad (56)$$

#### 6.1.2 Counting Functions

Let  $M$  be a set with  $m$  elements and  $N$  a set with  $n$  elements. Since a function represents a choice of one of the  $n$  elements of the codomain for each of the  $m$  elements in the domain, the product rule says that the number  $a$  of functions from set  $M$  to  $n$  is

$$a = n^m \quad (57)$$

#### 6.1.3 The Sum Rule

Let  $A$  and  $B$  2 tasks with  $n_1$  and  $n_2$  ways to do  $A$  and  $B$  respectively, with the fact that none of the set of  $n_1$  ways is the same as any of the set of  $n_2$  ways. Then the number of ways to do the 2 tasks is

$$n = n_1 + n_2 \quad (58)$$

#### 6.1.4 Sum Rule with Sets

The sum rule can be expressed in terms of sets. Let  $A$  and  $B$  be 2 disjoint sets. Then

$$|A \cup B| = |A| + |B| \quad (59)$$

This is the same as the inclusion-exclusion principle but the factor  $|A \cap B| = \emptyset$  because there are disjoint. So more generally, for  $n$  sets disjoint

$$|A_1 \cup \dots \cup A_n| = |A_1| + \dots + |A_n| \text{ when } A_i \cap A_j = \emptyset \forall i, j \quad (60)$$

### 6.2 Pigeonhole Principle

#### 6.2.1 Definition

If  $k$  is a positive integer and  $k + 1$  objects are placed into  $k$  boxes, then there are 2 or more objects in at least 1 box. The proof can be made by contraposition

- Suppose none of the  $k$  boxes has more than one object.
- Then the total number of objects would be at most  $k$ .
- This contradicts the assumption that we have  $k + 1$  objects.

### 6.2.2 Generalization of Pigeonhole Principle

If  $N$  objects are placed into  $k$  boxes, then there is at least 1 box that contains at least  $\lceil \frac{N}{k} \rceil$  objects. The proof can be made by contraposition

- Suppose none of boxes contains more than  $\lceil \frac{N}{k} \rceil - 1$  objects.
- Since  $\lceil \frac{N}{k} \rceil < \frac{N}{k}$ , the total number of objects is at most

$$k(\lceil \frac{N}{k} \rceil - 1) < k((\lceil \frac{N}{k} \rceil + 1) - 1) = N$$

- This contradicts the assumption that we have  $N$  objects.

### 6.2.3 Example with Functions

A function  $f$  from a set with  $k + 1$  elements to a set with  $k$  elements cannot be one-to-one.

- Create a box for each element  $y$  in the codomain of  $f$ .
- Put in the box for  $y$  all the elements  $x$  from the domain such that  $f(x) = y$ .
- Because there are  $k + 1$  elements and only  $k$  boxes, at least one box has two or more elements.
- Hence, the function cannot be one-to-one.

### 6.2.4 Example with Months

How many people among 100 are born in the same months ? Using the pigeon-hole principle, there are  $\lceil \frac{100}{12} \rceil$  people born in the same month.

## 6.3 Permutation and Combination

### 6.3.1 Definition of Permutation

A permutation of a set of distinct objects is an ordered arrangement of these objects. An ordered arrangement of  $r$  elements is called a  $r$ -permutation. The number of  $r$ -permutation  $p$  of a set with  $n$  elements is denoted by

$$p = P(n, r) \tag{61}$$

### 6.3.2 Counting Number of Permutation

If  $n$  is a positive integer and  $r$  is an integer with  $1 \leq r \leq n$  then

$$P(n, r) = \frac{n!}{(n - r)!} \tag{62}$$

### 6.3.3 Definition of Combination

An  $r$ -combination of elements of a set is an unordered selection of  $r$ -elements of a set. Thus, an  $r$ -combination is simply a subset of set with  $r$ -elements. By definition

$$C(n, r) = \binom{n}{r} = \frac{n!}{r!(n-r)!} \quad (63)$$

We see that we have a factor  $r!$  in the denominator of the fraction, and not in the permutation. That's because in combination, we does not want order. So for example the set  $\{1, 2, 2\} = \{2, 1, 2\}$ . So we "remove" these possibilities.

### 6.3.4 Example with Cards

Let say that we have a set of poker cards, so 52. How many hands of 5 cards can we made with them ? Here it's a combination because it's **unordered**. So here it's  $C(52, 5) = \frac{52!}{5!47!} = 2.5$  millions.

## 6.4 Binomial Coefficients and Identities

### 6.4.1 Binomial Theorem

#### Binomial Theorem

$$(x + y)^n = \sum_{j=0}^n \binom{n}{j} x^{n-j} y^j = \binom{n}{0} x^n + \binom{n}{1} x^{n-1} y + \dots + \binom{n}{n} y^n \quad (64)$$

### 6.4.2 Proof of Binomial Theorem

We use a combinational reasoning. The terms in the expansion of  $(x+y)^n$  are of the form  $x^{n-j}y^j$  for  $j = 0, 1, \dots, n$ . To form these terms, it's necessary to choose  $n-j$  times an  $x$  from the  $n$  sums. Therefore, the coefficient of  $x^{n-j}y^j$  is  $\binom{n}{n-j}$  which equal to  $\binom{n}{j}$

### 6.4.3 Exponential Identity

With  $n \geq 0$ , we have that

$$2^n = \sum_{j=0}^n \binom{n}{j} \quad (65)$$

That proof is really simple

$$2^n = (1 + 1)^n = \sum_{k=0}^n \binom{n}{k} 1^k 1^{n-k} = \sum_{k=0}^n \binom{n}{k}$$



#### 6.4.4 Pascal's Identity

If  $n, k$  are integers with  $n \geq k \geq 0$  then

$$\binom{n}{k-1} + \binom{n}{k} = \binom{n+1}{k} \quad (66)$$

The proof is fully algebraic

$$\begin{aligned} \binom{n}{k-1} + \binom{n}{k} &= \frac{n!}{(k-1)!(n-k+1)!} + \frac{n!}{k!(n-k)!} = \frac{n!k}{k!(n-k+1)!} + \frac{n!(n-k+1)}{k!(n-k+1)!} = \\ &= \frac{n!(n-k+1)}{k!(n-k+1)!} = \frac{(n+1)!}{k!(n-k+1)!} = \binom{n+1}{k} \end{aligned}$$

### 6.5 Permutation and Combination with Repetitions

#### 6.5.1 Permutation with Repetitions

An  $r$ -permutation with repetitions of a set of distinct objects is an ordered arrangement of  $r$ -elements from a set, where elements can occur multiple times. The number  $p$  of  $r$ -permutations of a set of  $n$  objects with repetitions allowed is

$$p = n^r \quad (67)$$

#### 6.5.2 Example of r-Permutation with Repetitions

How many strings of length  $r$  can be formed from the upper-case letters in the English alphabet? It's  $26^r$

#### 6.5.3 Combination with Repetitions

An  $r$ -combination with repetitions of elements of a set is an unordered selection of  $r$ -elements from a set, where elements can occur multiple times. The number of  $r$ -combinations from a set with  $n$  elements when repetitions is allowed is

$$C(n+r-1, r) = C(n+r-1, n-1) \quad (68)$$

#### 6.5.4 Example of r-Combination with Repetitions

How many solutions does the equation  $x_1 + x_2 + x_3 = 11$  have, where the  $x_i$  are non-negative integers. Each solution corresponds to a way to select 11 items from a set with 3 elements:  $x_1$  elements of type 1,  $x_2$  elements of type 2 and  $x_3$  of type 3. By the definition of combination with repetitions, we have  $C(3+11-1, 11) = C(13, 11) = C(13, 2) = \frac{13!}{11!2!} = 78$ .

### 6.5.5 Permutations with Indistinguishable Objects

The number  $p$  of different permutations of  $n$  objects, where there are  $n_1$  indistinguishable objects of type 1,  $n_2$  indistinguishable objects of type 2,..., and  $n_k$  indistinguishable objects of type  $k$  is

$$p = \frac{n!}{n_1!n_2!\dots n_k!} \quad (69)$$

## 7 Advanced Counting

### 7.1 Counting with Recurrence Relations

#### 7.1.1 Recurrence Relation

A recurrence relation for a sequence  $\{a_n\}$  is an equation that express  $\{a_n\}$  in terms of a finite number  $k$  of the preceding terms of the sequence  $\Leftrightarrow a_n = f(a_{n-1}, \dots, a_{n-k})$ . If its terms satisfy the recurrence, it's a solution. The first terms are the initial conditions. Example:

$$f_0 = 0, f_1 = 1 \text{ and } f_n = f_{n-1} + f_{n-2}$$

#### 7.1.2 Counting Bit Strings

Find a recurrence relation and give initial conditions for the number of bit strings of length  $n$  without two 0 consecutive.

- Let  $A_n$  denotes the bit strings of length  $n \geq 3$ . without two consecutive 0s and let  $s_n = |A_n|$ .
- A bit string  $s_n \in A_n$  can end with a 1 or a 0, ie.  $s_n = s_{n-1}1$  or  $s_n = s_{n-1}0$ .
- If  $s_n$  ends in a 1,  $s_n = s_{n-1}$ , where  $s_{n-1}$  can be any bit strings  $\in A_{n-1}$ . If  $s_n$  ends in a 0,  $s_n$  has to end with a 1, thus  $s_{n-1} = s_{n-2}1$  where  $s_{n-2}$  can be any bit strings  $\in A_{n-2}$ .  $a_n = a_{n-1} + a_{n-2}$ .
- Therefore,  $a_n = a_{n-1} + a_{n-2}$

### 7.2 Solving Linear Recurrence Relations

#### 7.2.1 Linear Homogeneous Recurrence Relation

A linear homogeneous recurrence relation of degree  $k$  with constant coefficients is a recurrence relation of the form

$$a_n = c_1 a_{n-1} + c_2 a_{n-2} + \dots + c_k a_{n-k} \quad (70)$$

where  $c_1, \dots, c_k \in \mathbb{R}^*$ .

### 7.2.2 Examples of Linear Homogeneous Recurrence Relation

- $P_n = (1.2)P_{n-1}$  is a linear homogeneous recurrence relation of degree 1.
- $a_n = a_{n-1}^2$  is not linear.
- $H_n = 2H_{n-1} + 1$  is not homogeneous (the +1 is not a multiple of previous terms).

### 7.2.3 Characteristic Equation

Let  $a_n = c_1a_{n-1} + c_2a_{n-2} + \dots + c_ka_{n-k}$  and assume  $a_n = r^n$ , with  $r \in \mathbb{R}$ . If we replace  $a_i$  by  $r^{n-i}$ , we obtain the characteristic equation

$$r^k - c_1r^{k-1} - \dots - c_{k-1}r - c_k = 0 \quad (71)$$

### 7.2.4 Solving Linear Homogeneous Recurrence Relation of degree 2

Let  $c_1$  and  $c_2$  be real numbers and that  $r^2 - c_1r - c_2 = 0$  has 2 different roots  $r_1$  and  $r_2$ . Then the sequence  $\{a_n\}$  is a solution to the recurrence relation  $a_n = c_1a_{n-1} + c_2a_{n-2}$  if and only if  $a_n = \alpha_1r_1^n + \alpha_2r_2^n$  where  $\alpha_i \in \mathbb{R}$ .

### 7.2.5 Example of solving Linear Homogeneous Recurrence Relation of degree 2

Let  $a_n = a_{n-1} + 2a_{n-2}$  with  $a_0 = 2$  and  $a_1 = 7$ . So the characteristic equation is  $r^2 - r - 2 = 0$  with roots  $r_1 = 2$  and  $r_2 = -1$ . Therefore,  $\{a_n\}$  is a solution to the recurrence relation if and only if  $a_n = \alpha_12^n + \alpha_2(-1)^n$ . We resolve this equation with the initial conditions and so we found  $\alpha_1 = 3$  and  $\alpha_2 = -1$  and so  $a_n = 3 * 2^n - (-1)^n$ .

If there are repeated roots, for example the only root is 3 but with multiplicity 2, then the equation to solve is  $a_n = \alpha_13^n + \alpha_2n3^n$ .

### 7.2.6 Solving Linear Homogeneous Recurrence Relation of Arbitrary degree

Let  $c_1, \dots, c_k$  be real numbers and that  $r^k - c_1r^{k-1} - \dots - c_k = 0$  has  $k$  different roots  $r_1, \dots, r_k$ . Then the sequence  $\{a_n\}$  is a solution to the recurrence relation  $a_n = c_1a_{n-1} + c_2a_{n-2} + \dots + c_ka_{n-k}$  if and only if  $a_n = \alpha_1r_1^n + \alpha_2r_2^n + \dots + \alpha_kr_k^n$  where  $\alpha_i \in \mathbb{R}$ .

## 7.3 Generating Functions

### 7.3.1 Definition

The generating function of the infinite sequence  $a_0, \dots, a_k, \dots$  of real numbers is the infinite series

$$G(x) = \sum_{k=0}^{\infty} a_k x^k \quad (72)$$

### 7.3.2 Examples of Generating functions

- The sequence  $\{a_k\}$  with  $a_k = 3$  has the generating function  $\sum_{k=0}^{\infty} 3x^k$
- The sequence  $\{a_k\}$  with  $a_k = k + 1$  has the generating function  $\sum_{k=0}^{\infty} (k + 1)x^k$

### 7.3.3 Solving Recurrence Relation with Generating Functions

Solve the recurrence relation  $a_k = 3a_{k-1}$  with  $a_0 = 2$ .

- $G(x) = \sum_{k=0}^{\infty} a_k x^k$
- $xG(x) = \sum_{k=0}^{\infty} a_k x^{k+1} = \sum_{k=1}^{\infty} a_{k-1} x^k$
- $G(x) - 3xG(x) = \sum_{k=0}^{\infty} a_k x^k - 3 \sum_{k=1}^{\infty} a_{k-1} x^k = a_0 + \sum_{k=1}^{\infty} (a_k - 3a_{k-1}) x^k = 2$
- $G(x) - 3xG(x) = (1 - 3x)G(x) = 2$
- $G(x) = \frac{2}{1-3x}$  and  $\frac{1}{1-ax} = \sum_{k=0}^{\infty} a^k x^k$
- $G(x) = 2 \sum_{k=0}^{\infty} 3^k x^k = \sum_{k=0}^{\infty} 2 * 3^k x^k$
- So  $a_n = 2 * 3^k$ .

## 7.4 Counting Problem

### 7.4.1 Extended Binomial Coefficients

$$\binom{u}{k} = \begin{cases} \frac{u(u-1)\dots(u-k+1)}{k!} & \text{if } k > 0 \\ 1 & \text{if } k = 0 \end{cases} \quad (73)$$

### 7.4.2 Extended Binomial Theorem

Let  $x \in \mathbb{R}$  with  $|x| < 1$  and let  $u$  be a real number. Then

$$(1+x)^u = \sum_{k=0}^n \binom{u}{k} x^k \quad (74)$$

## 7.5 Inclusion-Exclusion

### 7.5.1 Inclusion-Exclusion Principle (Recall)

#### Inclusion-Exclusion Principle

$$|A_1 \cup \dots \cup A_n| = \sum_i^n |A_i| - \sum_{1 \leq i < k \leq n} |A_i \cap A_k| + \sum_i^n |A_i| - \sum_{1 \leq i < j < k \leq n} |A_i \cap A_k \cap A_j| + \dots + (-1)^n \left| \bigcap_i^n A_i \right|$$

### 7.5.2 Derangements

A derangement is a permutation of objects that leaves no objects in the original position.

#### Number of Derangements with n elements

$$D_n = n! \left[ 1 - \frac{1}{1!} + \frac{1}{2!} - \frac{1}{3!} + \dots + (-1)^n \frac{1}{n!} \right]$$

The proof is not tricky but it works with the inclusion-exclusion principle.

Let  $P_i$  be the set of permutations that fix element  $i$ . The set of permutations that are not derangements is then  $P_1 \cup P_2 \cup \dots \cup P_n$ . And so the number of derangements is  $n! - (P_1 \cup P_2 \cup \dots \cup P_n)$ .

- Now  $|P_i| = (n-1)!$ ,  $|P_i \cap P_k| = (n-2)!$  for  $i \neq k$  and in general  $|\bigcup_{i \in \mathcal{I}} P_i| = (n-s)!$  if  $s = |\mathcal{I}|$ .
- Using the inclusion-exclusion principle:  $|P_1 \cup P_2 \cup \dots \cup P_n| = \sum_{1 \leq i \leq n} |P_i| - \sum_{1 \leq i < k \leq n} |P_i \cap P_k| + \dots + (-1)^n \left| \bigcap_{1 \leq i \leq n} P_i \right| = \binom{n}{1}(n-1)! - \binom{n}{2}(n-2)! + \dots + (-1)^n \binom{n}{n}$
- Since  $\binom{n}{k}(n-k)! = \frac{n!}{k!}$ ,  $D_n = n! - \frac{n!}{1} + \frac{n!}{2} - \dots + (-1)^n \frac{n!}{n!} = n! \left[ 1 - \frac{1}{1!} + \frac{1}{2!} - \frac{1}{3!} + \dots + (-1)^n \frac{1}{n!} \right]$ .

## 8 Probability

### 8.1 Introduction to Probability

#### 8.1.1 Probability of an Event

If  $S$  is a finite sample space of equally likely outcomes, and  $E$  is an event, so a subset of  $S$ , then the probability of the event  $E$  is

$$p(E) = \frac{|E|}{|S|} \quad (75)$$

We can express this with the different outcomes of the event

$$p(E) = \sum_{s \in E} p(s) \quad (76)$$

The probability of an event  $E$  is always bounded by 0 and 1

$$0 \leq p(E) \leq 1 \quad (77)$$

#### 8.1.2 Probability of Complement of Events

Let  $E$  an event on a sample space  $S$ . Then the complement of  $E$  is denoted by  $\bar{E} = S - E$  and its probability is

$$p(\bar{E}) = 1 - p(E) \quad (78)$$

#### 8.1.3 Probability of Union of Events

Let  $E_1$  and  $E_2$  events on a sample space  $S$ . Then the probability of the union of  $E_1$  and  $E_2$  is

$$p(E_1 \cup E_2) = p(E_1) + p(E_2) - p(E_1 \cap E_2) \quad (79)$$

The proof is a direct proof by the inclusion-exclusion principle.

#### 8.1.4 Independence of Events

##### Independence of Events

Let  $E$  and  $F$  be events. Then we say that  $E$  and  $F$  are independent if and only if

$$p(E \cap F) = p(E)p(F) \quad (80)$$

## 8.2 Probability Theory

### 8.2.1 Limitation of Laplace's Definition

This limitation informs that not all events are equally likely. It means that some events can have a greater probability to appear than others, like make a 6 with 2 dice than a 2.

### 8.2.2 Probability Distribution

Let  $S$  a sample space of an experiment with a finite or countable number of outcomes. We assign a probability  $p(s)$  to each element  $s$  of  $S$ . Then

$$\sum_{s \in S} p(s) = 1 \quad (81)$$

### 8.2.3 Combination of Events

If  $E_1, \dots, E_k$  is a sequence of pairwise disjoint events on a sample space  $S$ , then

$$p\left(\bigcup_i E_i\right) = \sum_i p(E_i) \quad (82)$$

### 8.2.4 Uniform Distribution

Suppose that  $S$  is a set with  $n$  elements. The uniform distribution assigns to each element a probability  $\frac{1}{n}$ . For example tossing a fair coin with probability  $\frac{1}{2}$

### 8.2.5 Pairwise and Mutual Independence

The events  $E_1, \dots, E_n$  are pairwise independent if and only if

$$p(E_i \cap E_j) = p(E_i)p(E_j) \quad \forall i, j \quad (83)$$

The events are said mutually independent if

$$p(E_{i_1} \cap E_{i_2} \cap \dots \cap E_{i_m}) = p(E_{i_1})p(E_{i_2}) \dots p(E_{i_m}) \quad (84)$$

whenever  $i_j, j = 1, \dots, m$  are integers with  $1 \leq i_1 < i_2 < \dots < i_m \leq n$  and  $m \geq 2$ .

Mutual independence implies pairwise independence, but not the opposite.

### 8.2.6 Independent Bernoulli Trials

The probability of  $k$  successes in  $n$  independent Bernoulli Trials, so an event with only 2 possibilities, with probability of success  $p$  and probability of failures  $q = 1 - p$  is

$$C(n, k)p^k q^{n-k} \quad (85)$$

## 8.3 Conditional Probability

### 8.3.1 Definition

The conditional of a set  $E$  given a set  $F$ , denoted by  $p(E|F)$ , is defined as

$$p(E|F) = \frac{p(E \cap F)}{p(F)} \text{ or written } \frac{p(E, F)}{p(F)} \quad (86)$$

If  $E$  and  $F$  are independent, then  $p(E|F) = p(E)$

### 8.3.2 Bayes' Theorem

Suppose that  $E$  and  $F$  are events from a sample space  $S$  such that  $p(E) \neq 0$  and  $p(F) \neq 0$  then

$$p(F|E) = \frac{p(E|F)p(F)}{p(E)} = \frac{p(E|F)p(F)}{p(E|F)p(F) + p(E|\bar{F})p(\bar{F})} \quad (87)$$

### 8.3.3 Generalization of Bayes' Theorem

Suppose that  $E$  is an event from a sample space  $S$  and that  $F_1, \dots, F_n$  are mutually exclusive events such that  $\bigcup_i^n F_i = S$ . Assume also that  $p(E) \neq 0$  for  $i = 1, 2, \dots, n$ . Then

$$p(F_j|E) = \frac{p(E|F_j)p(F_j)}{\sum_{i=1}^n p(E|F_i)p(F_i)} \quad (88)$$



## 9 Advanced Probability

### 9.1 Expected Value

#### 9.1.1 Random Variable

A random variable  $X : S \rightarrow \mathbb{R}$  is a function from a sample space  $S$  of an experiment to the set of real numbers. It assigns an  $\mathbb{R}$ -value to each possible outcomes. It's a function but not random ! For example, we can define a random variable  $X(s)$  for  $s \in \{H, T\}$ , the different outcomes when we toss a fair coin. We can define that  $X(H) = 1$  and  $X(T) = 0$ .

#### 9.1.2 Distribution of Random Variable

The distribution of a random variable on a sample space  $S$  is the set of pairs  $(r, p(X = r))$  for all  $r \in X(S)$ , where  $p(X = r)$  is the probability that  $X$  takes the value  $r$ .

$$p(X = r) = \sum_{s \in S: X(s)=r} p(s) \quad (89)$$

#### 9.1.3 Expected Value

The expected value of a random variable  $X$  on a sample space  $S$  is equal to

$$E(X) = \sum_{s \in S} p(s)X(s) \quad (90)$$

The basic example of this is the expected value of a fair dice. The outcomes are  $\{1, 2, 3, 4, 5, 6\}$  and each has probability  $\frac{1}{6}$ . So  $E(X) = 1 * \frac{1}{6} + 2 * \frac{1}{6} + 3 * \frac{1}{6} + 4 * \frac{1}{6} + 5 * \frac{1}{6} + 6 * \frac{1}{6} = \frac{7}{2}$ .

$E(XE(Y)) = E(X)E(Y)$  because  $E(X)$  is just a number !

#### 9.1.4 Expected Value for Bernoulli Trials

##### Expected Value for Bernoulli Trials

The expected number of successes when  $n$  mutually independent Bernoulli Trials are performed, where  $p$  is the probability of each trail, is **np**

The proof follows the definition of expected value

$$\begin{aligned} E(X) &= \sum_{k=1}^n kp(X = k) = \sum_{k=1}^n kC(n, k)p^k q^{n-k} = \sum_{k=1}^n nC(n-1, k-1)p^k q^{n-k} \\ &= np \sum_{k=1}^n C(n-1, k-1)p^{k-1} q^{n-k} = np \sum_{j=1}^{n-1} C(n-1, j)p^j q^{n-1-j} = np(p+q)^{n-1} = np. \end{aligned}$$

### 9.1.5 Linearity of Expected Value

If  $X_i$ ,  $i = 1, \dots, n$  with  $n$  as positive integer, are random variables on a sample space  $S$ , then

- $E(X_1 + \dots + X_n) = E(X_1) + \dots + E(X_n)$
- $E(aX + b) = aE(x) + b$

The proof follows the definition of expected value

$$\begin{aligned} \bullet E(X_1 + X_2) &= \sum_{s \in S} p(s)(X_1(s) + X_2(s)) = \sum_{s \in S} p(s)X_1(s) + \sum_{s \in S} p(s)X_2(s) = E(X_1) + E(X_2) \\ \bullet E(aX + b) &= \sum_{s \in S} p(s)(aX(s) + b) = a \sum_{s \in S} p(s)X(s) + b \sum_{s \in S} p(s) = aE(X) + b \end{aligned}$$

### 9.1.6 Independent Random Variables

The random variable  $X$  and  $Y$  on a sample space  $S$  are independent if and only if

$$p(X = r_1 \wedge Y = r_2) = p(X = r_1)p(Y = r_2) \quad (91)$$

Also, if  $X$  and  $Y$  are independent, then

$$E(XY) = E(X)E(Y) \quad (92)$$

## 9.2 Variance

### 9.2.1 Definition

let  $X$  be a random variable on a sample space  $S$ . The variance of  $X$ , denoted by  $V(X)$  is

$$V(X) = \sum_{s \in S} (X(s) - E(X))^2 p(s) \quad (93)$$

### 9.2.2 Standard Deviation

The standard deviation of a random variable  $X$ , denoted by  $\sigma(X)$  is

$$\sigma(X) = \sqrt{V(X)} \quad (94)$$

Variance and standard deviation are used to quantify how widely a random variable is distributed.

### 9.2.3 Example to explain the variance

Let  $S$  be a sample space with elements  $\{1, 2, 3, 4, 5, 6\}$ . let  $X(s) = 0$  for all  $s \in S$ . Let  $Y(s) = -1$  for the 3 first elements and  $Y(s) = 1$  for the 3 last.  $E(X) = E(Y) = 0$ . But here  $V(X) = 0$  and  $V(Y) = 1$ .

#### 9.2.4 Characterization of Variance

$$V(X) = E(X^2) - E(X)^2 = E((X - E(X))^2) \quad (95)$$

#### 9.2.5 Variance for Bernoulli Trials

Let  $X$  a random variable with  $X(s) = 1$  if it's a success and  $X(s) = 0$  if it's a failure, and where  $p$  is the probability of success and  $q$  of failure. Because  $X$  can only take value 0 or 1,  $X^2(x) = X(s)$ . Hence

$$V(X) = pq \quad (96)$$

#### 9.2.6 Variance for Independent Random Variables

If  $X_i$ ,  $i = 1, \dots, n$  with  $n$  as positive integer, are pairwise independent random variables on a sample space  $S$ , then

$$V(X_1 + \dots + X_n) = V(X_1) + \dots + V(X_n) \quad (97)$$

### 9.3 Estimation Deviation

#### 9.3.1 Markov Inequality

Let  $X$  be a non-zero and non-negative random variable. Let  $p(X \geq a)$  denotes the probability that the variable attains a value greater than  $a$ . Then

$$\forall M > 0 \quad p(X \geq ME(X)) \leq \frac{1}{M} \quad (98)$$

#### 9.3.2 Chebyshev's Inequality

Let  $X$  be a random variable on a sample space  $S$  with probability function  $p$ . If  $r \in \mathbb{R}$  then

$$p(|X(s) - E(X)| \geq r) \leq \frac{V(X)}{r^2} \quad (99)$$

### 9.4 Geometrical Distribution

#### 9.4.1 Definition

The geometric distribution describes the probability of experiencing a certain amount of failures before experiencing the first success in a series of Bernoulli trials (2 possibilities, for example tossing a coin  $n$  times).

If a random variable  $X$  follows a geometric distribution, then the probability of experiencing  $k$  failures before experiencing the first success can be found by the following formula

$$p_X(k) = (1 - p)^k p \quad (100)$$

where  $k$  is the number of failures and where  $p$  is the probability of successes for each trial.

#### 9.4.2 Example of Geometrical Distribution

What is the expected number of flips till a tail show up ?

First, let's define the probability to have a tail:  $p(T) = p$  and the probability to have a head:  $p(H) = 1 - p$ . With that, we have an infinite sample space  $S : T, HT, HHT, \dots$ . We see directly that tossing coins follows the geometrical distribution:

$$p(T) = p \text{ so } p(HT) = (1 - p)p \text{ so } p(H^k T) = (1 - p)^k p \quad (101)$$

Let's now define a random variable  $X(S) = \text{number of flips to get a T}$ . We have now  $p_X(j) = (1 - p)^{j-1} p$ . Now we can compute the  $E(X)$  with the definition.

$$E(X) = \sum_{j=1}^{\infty} j p_X(j) = \sum_{j=1}^{\infty} j (1 - p)^{j-1} p = p \sum_{j=1}^{\infty} j (1 - p)^{j-1} \quad (102)$$

But we know that

$$\sum_{j=1}^{\infty} j x^{j-1} = \frac{1}{(1 - x)^2} \quad (103)$$

So now we can compute:

$$p \frac{1}{(1 - 1 + p)^2} = \frac{p}{p^2} = \frac{1}{p} \quad (104)$$

So the expected value becomes  $\frac{1}{p}$

## 10 Entropy and Informations

### 10.1 Symbols

#### 10.1.1 Informations and symbols

A symbol  $S_i$  provides informations if and only if it is a random variable non-trivial (can take more than one value) over an Alphabet called  $A$ . For example,  $A$  can be  $[0, 1]$  for tossing a coin.

$$S_i = \Omega \rightarrow A \quad (105)$$

### 10.2 Amount of Informations

#### 10.2.1 Conventions

- $0 * \log_b(0) = 0$  because  $\lim_{\epsilon \rightarrow 0} \epsilon \log_b(\epsilon) = 0$

#### 10.2.2 Hartley's Measure

The amount of informations for Hartley by a symbol of an Alphabet  $A$  is

$$Amount = \log |A| \quad (106)$$

But there is a problem. For example with the weather, if we want say that all days in this week will be sunny, and the last rainy with 0 for rainy and 1 for sunny ; we will have more informations in this message than if we announce that it will rain 3 days, and sun 4 days ; and that's not correct with Hartley description because rainy and sunny are in the same Alphabet here !

#### 10.2.3 Shannon's Theory (Entropy)

The entropy  $H(S)$ , associated to a random variable  $S$ , is a mathematical function to, like Hartley's Measure, measure the amount of informations contained in a source of informations.

$$H_b(S) = - \sum_{s \in Supp(p_S)} p_S(s) \log(p_S(s)) \quad (107)$$

where  $Supp(p_S)$  means without zero-probability events. It's needed to avoid  $\log_b(0)$ . Because by convention we say that  $p_S(s) \log(p_S(s)) = 0$  when  $p_S(s) = 0$ , we can write

$$H_b(S) = - \sum_{s \in A} p_S(s) \log(p_S(s)) \quad (108)$$

The  $b$  defines the unit. For bits, we say that  $b = 2$ . Now we can think to evaluate this by first computing  $-\log(p_S(s))$  for each  $s \in A$ , and then take the average (excluding zero-probability terms).

$$H(S) = E[-\log(p_S(s))] \quad (109)$$

We see now that the 2 measures are the same when the random variable is uniformly distributed over the Alphabet. When it is, we have  $p_S(s) = \frac{1}{|A|}$  and  $-\log(p_S(s)) = \log(|A|)$ , which is constant. In this case,  $H(S) = E(\log(|A|)) = \log(|A|)$ .

So to summarize, we can compute the entropy of a source of information with this:

#### Entropy Calculation

$$H_b(X) = -E[\log_b P(X)] = \sum_{i=1}^n P_i \log_b\left(\frac{1}{P_i}\right) = -\sum_{i=1}^n P_i \log_b(P_i) \quad (110)$$

#### 10.2.4 Binary Entropy Function

Let's take  $S \in [0, 1]$  be a binary random variable.  $p_S(s)$  is described by one parameter, so  $p = p_S(s)$ .

$$H_2(S) = -[p \log_2(p) + (1 - p) \log_2(1 - p)] = h(p) \quad (111)$$

Where  $h(p)$  is called **the binary entropy function**. For  $p = 0$ ,  $h(p) = 0$ . For  $p = 1$ ,  $h(p) = 0$ . For  $p = \frac{1}{2}$ ,  $h(p) = 1$ .

### 10.3 Applications and Theorems

#### 10.3.1 Theorem 1 (Entropy Bound)

Let  $S \in A$  be a random variable, then

#### Entropy Bound

$$0 \leq H_b(S) \leq \log_b |A| \quad (112)$$

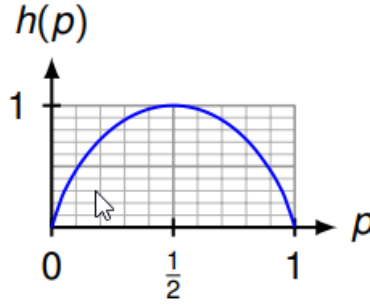


Figure 13: Graph of the Binary Entropy Function

The first inequality becomes an equality if  $S$  is constant ; and the second too if  $S$  is uniformly distributed. The proof is really important because it's the base of the entropy comprehension

- First for the left inequality, we observe that  $-p(s) \log(p(s))$  is 0 if  $p(s) \in \{0, 1\}$  and  $> 0$  if  $0 < p(s) < 1$ . Hence  $H(S) \geq 0$  with equality if and only if  $p(s) \in \{0, 1\}$  for all values.
- To prove the right inequality, we use a trick that often works in inequalities involving entropies: to prove that  $A \leq B$ , we prove that  $A - B \leq 0$ .
- So  $H(S) - \log |A| = E(-\log(p(s))) - \log |A| = E(-\log(p(s)) - \log |A|) = E(\frac{1}{p(s)|A|})$ .
- This equals so  $\sum_{s \in A} p(s) [\log(\frac{1}{p(s)|A|})] \leq \sum_{s \in A} p(s) [\log(\frac{1}{p(s)|A|}) - 1] \log(e) = \log(e) \sum_{s \in A} [\frac{1}{|A|} - p(s)] = \log(e)(1 - 1) = 0$ .
- And with equality if and only if  $p(s)|A| = 1$  for all  $s$ .

We use the **IT-Inequality** that says for  $r > 0$

$$\log_b(r) \leq (r - 1) \log_b(e) \quad (113)$$

And equality if  $r = 1$

### 10.3.2 Entropy with more than 1 random variable

Let  $X : \Omega \rightarrow \Lambda$  and  $Y : \Omega \rightarrow \Upsilon$  be random variables, then

### Entropy with Multiple Random Variables

$$H_b(X, Y) = - \sum_{x, y \in \Lambda_X \Upsilon} p_{X, Y}(x, y) \log(p_{X, Y}(x, y)) \quad (114)$$

$$H_b(X, Y) = E[-\log_b(p_{X, Y}(x, y))] \quad (115)$$

$$H_b(X, Y) = - \sum_i p_i \log(p_i) \quad (116)$$

#### 10.3.3 Notations

- $S_1, S_2, \dots, S_n = [S_i]_{i=1}^n$

#### 10.3.4 Example with many random variables

Let  $[S_i]_{i=1}^n$  be coin flips  $S_i \in [0, 1] = A$ . If the coin is fair, and so all flips are independent, then  $p_{S_1, S_2, \dots, S_n}(s_1, s_2, \dots, s_n) = \prod_{i=1}^n p_{S_i}(s_i) = (\frac{1}{2})^n$  because all flips have probability one half, and this  $\forall ([S_i]_{i=1}^n) \in [0, 1]^n$

So  $H_2([S_i]_{i=1}^n) = \log_2|[0, 1]^n| = n$  bits. It's logic because we must have only 1 number, 0 or 1, to send the information of the result. It's equal to the logarithm of the cardinality of the set because it's uniformly distributed.

#### 10.3.5 Theorem 2 (Addition of Entropies)

Let  $[S_i]_{i=1}^n$  be random variables, then

$$H([S_i]_{i=1}^n) \leq H(S_1) + H(S_2) + \dots + H(S_n) \quad (117)$$

The inequality becomes an equality if all random variables are independent.

### 10.4 Conditional Entropy

#### 10.4.1 Global definition

The conditional entropy of a random variable X given Y is defined as



### Conditional Entropy

$$H(X|Y) = \sum_{y \in Y} p_Y(y) H(X|Y = y) \quad (118)$$

$$H(X|Y) = -E[\log_{X|Y}(x|y)] \quad (119)$$

$$H(X|Y) = - \sum_{y \in Y} p_{X,Y}(x, y) \log\left(\frac{p_{X,Y}(x, y)}{p_Y(y)}\right) \quad (120)$$

#### 10.4.2 Theorem 5 (Inequalities)

Let  $X$  and  $Y$  random variables, then

$$H(X|Y) \leq H(X) \quad (121)$$

The inequality becomes an equality if  $X$  and  $Y$  are independent.

#### 10.4.3 Theorem 6 (Change Rule of Entropy)

Let  $[S_i]_{i=1}^n$  be random variables, then

$$H(S_1, \dots, S_n) = H(S_1) + \sum_{i=2}^n H(S_i | S_1, \dots, S_{i-1}) \quad (122)$$

#### 10.4.4 Notes About Entropy

- Order does not matter:  $H(X, Y, Z) = H(Z, X, Y)$
- $H(X|Y) = H(X, Y) - H(Y)$

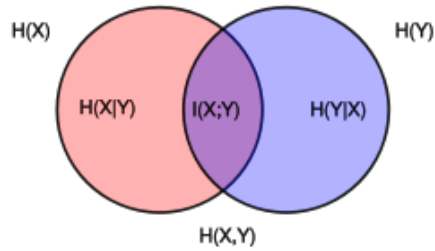


Figure 14: Representation of Entropy with Sets

## 11 Source Coding

### 11.1 Basics of Source Coding Theory

#### 11.1.1 Schema of transmission

A source, for example a mail box, sends symbols  $S_i \in A$  to a source encoder that owns an encoding map  $\Gamma : A \rightarrow C$  that maps every symbols from an input alphabet  $A$  to a codebook  $C$  with a function one-to-one. With that we have a codebook  $C = [\Gamma(s_1), \dots, \Gamma(S_n)]$  and an output alphabet  $D =$  all different part that can contains an output symbol.

#### 11.1.2 Basic example

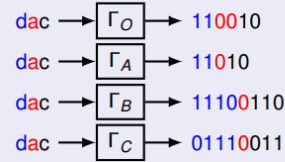
Let  $\Gamma : [a, b] \rightarrow [0, 11]$ . So we have  $D = [0, 1]$  and the word  $abba = 011110$ .

#### 11.1.3 Definitions

- A codebook  $C$  is uniformly decodable (UD) if every concatenation of code-words has a unique parsing.
- A codebook  $C$  is said to be prefix-free, also called instantaneous, if no codewords is the prefix of an other.
- We said that an output alphabet  $D$  is  $D$ -ary where  $D$  is the cardinality of  $D$ . For example if  $D = [0, 1, 2]$ , then  $D$  is a 3-ary output alphabet.

For each code, the encoding map  $\Gamma$  is specified in the following table:

$\mathcal{A}$	$\Gamma_O$	$\Gamma_A$	$\Gamma_B$	$\Gamma_C$
a	00	0	0	0
b	01	01	10	01
c	10	10	110	011
d	11	11	1110	0111



The source alphabet  $\mathcal{A}$ , the  $k$ , the code alphabet  $\mathcal{D}$  and the codebook  $\mathcal{C}$  are implicit from the encoding map.

Figure 15: Example of Encoding Map

#### 11.1.4 Theorem 1 (First Implications)

- If a codebook  $C$  is prefix-free  $\Rightarrow$  UD.
- If a codebook  $C$  is UD, it does **not** imply prefix-free.
- If a codebook  $C$  is reverse prefix-free (read the characters right-left)  $\Rightarrow$  UD.
- If a codebook  $C$  is reverse prefix-free it does not imply instantaneous.

### 11.2 Kraft-McMillan and Uniformly Decodable Codes

#### 11.2.1 Theorem 2 (Kraft-McMillan theorem Part 1)

If  $C$  is a  $D$ -ary code which is UD, its codeword lengths  $l_1, \dots, l_n$  satisfy

$$K(C) = \sum_{i=1}^n D^{-l_i} \leq 1 \quad (123)$$

Example:  $C = [00, 01, 10, 11]$   $D = 2$  then  $K(C) = 4 * 2^{-2} = 1 \leq 1$ .

#### 11.2.2 Theorem 2 (Kraft-McMillan Part 2)

Existence of Code Uniquely Decodable satisfying Kraft Inequality

If the positive integers  $l_1, \dots, l_n$  satisfy Kraft's inequality for some integers  $D$ , then  $\exists$  a  $D$ -ary prefix-free code (hence UD) that has codeword lengths  $l_1, \dots, l_n$ .

#### 11.2.3 Theorem 3 (Mix Implications with Theorem 1)

- $C$  is UD  $\Rightarrow K(C) \leq 1$
- $K(C) > 1 \Rightarrow$  not UD
- If the  $K(C) \leq 1$ , it does not imply that the codebook is UD ! We can only define that there  $\exists$  a  $D$ -ary prefix-free code (hence UD) that has codeword lengths  $l_1, \dots, l_n$  ; Theorem 2 from Kraft-MacMillan.

Kraft-McMillan Theorems implies that any uniquely decodable code can be substituted by a prefix-free code of the same codeword lengths.

## 11.3 Minimize Average Codewords Lengths

### 11.3.1 Explanation of Average Codewords Lengths

For a given source  $S : \Omega \rightarrow A$  and integers  $D = 2$ , find a  $D$ -ary code  $C$  for  $S$  and the corresponding encoding map  $\Gamma$  that minimize the average codeword lengths  $L(S, \Gamma)$

#### Definition of Average Codeword Lengths

$$L(S, \Gamma) = \sum_{s \in A} p_s l(\Gamma(s)) = \sum_{s \in A} p_s l(s) = \sum_i p_i l_i \quad (124)$$

### 11.3.2 Theorem 4 (Lower Bound for the Average Codeword Lengths)

Let  $\Gamma : A \rightarrow C$  be the encoding map of a  $D$ -ary code for  $S \in A$ . If the code is UD, then

$$H_D(S) \leq L(S, \Gamma) \quad (125)$$

The inequality becomes an equality if and only if  $\forall s \in A$

$$p_S(s) = D^{-l(s)} \text{ the same as } -\log(p_S(s)) = l(s) \quad (126)$$

- $H(S) - L(S, \Gamma) = \sum_i p_i \log(\frac{1}{p_i}) - \sum_i p_i \log(D^{l_i}) = \sum_i p_i \log(\frac{1}{p_i D^{l_i}})$
- This, by the IT-Inequality is  $\leq \log(e) \sum_i p_i (\log(\frac{1}{p_i D^{l_i}}) - 1) = \log(e) (\sum_i D^{-l_i} - \sum_i p_i = \log(e) \sum_i D^{-l_i} - 1 \leq 0$ .

## 11.4 Advanced Source Coding

For this, we assume that the symbols are not diadic, so that  $-\log(p(s))$  is not an integer for all  $s$  of a code.

### 11.4.1 Shannon-Fano Code

Shannon-Fano code is defined as a code with codewords length equal to  $l_i = \lceil -\log(p_i) \rceil$ , so the next integer. Is it full filled ? Yes, but not optimal !

$$\sum_{i=1}^M D^{l_i} = \sum_{i=1}^M D^{\lceil -\log(p_i) \rceil} \leq \sum_{i=1}^M D^{\log_D p_i} = \sum_{i=1}^M p_i = 1 \quad (127)$$

### 11.4.2 Huffman's Construction Code

It's a construction for a codebook but we start from the bottom of the tree. We place in decreasing order the probabilities of the symbols of the source and we

group the lowest probabilities to form a node with a probability define as the sum of the 2 orders ; and for the rest we do the same, until the top of the tree with probability 1. Then we define 0 and 1 for each branch and we have now the codewords for each symbols. And it's optimal and prefix free (so UD) !

We have now the inequality

$$H_2(S) \leq L(S, \Gamma_H) \leq L(S, \Gamma_{SF}) < H_2(S) + 1 \quad (128)$$

## 11.5 Generalization of Source Encoding

### 11.5.1 Different Types of Source

A source can be:

- $S$  a single random variable
- $S_1, S_2, \dots, S_n$  some finite random variables
- $S_1, \dots, S_n, \dots, \infty$  many random variables, also written  $\mathcal{S}$ .

### 11.5.2 Definition of Regular Source

The source  $\mathcal{S} = S_1, \dots, S_n, \dots$  is regular if:

- the entropy per symbol:  $H(\mathcal{S}) = \lim_{n \rightarrow \infty} H(S_n)$  exists and is finite
- the entropy rate:  $H^*(\mathcal{S}) = \lim_{n \rightarrow \infty} H(S_n | S_1, \dots, S_{n-1})$  exists and is finite

### 11.5.3 Inequality of Entropy Remake for many Sequences

For  $n$  sequences, we have

$$\frac{H(S_1, \dots, S_n)}{n} \leq \frac{L([S_1, \dots, S_n], \Gamma_H)}{n} < \frac{H(S_1, \dots, S_n)}{n} + \frac{1}{n} \quad (129)$$

Hence, if we encode  $n$  symbols at a time using an optimal D-ary code, as  $n \rightarrow \infty$ , the average codeword-length per source symbol tends to  $\frac{H(S_1, \dots, S_n)}{n}$ .

#### 11.5.4 Definition of Stationary Source

A source  $\mathcal{S} = S_1, \dots, S_n, \dots$  is stationary if for all positive integers  $n, k$  then  $\mathcal{S}$  has the same statistics (probability distribution) as  $(S_{1+k}, \dots, S_{n+k})$ . A source is stationary if its distribution is unaffected by an index shift (time shift). It means that for example:

- $p_{S_1} = p_{S_k}$
- $p_{S_1, S_2} = p_{S_k, S_{k+1}}$
- $p_{S_1, \dots, S_n}(s_1, \dots, s_n) = p_{S_{1+k}, \dots, S_{n+k}}(s_1, \dots, s_n), \quad \forall n, \quad \forall k,$   
 $\forall s_1, \dots, s_n \in \mathcal{A}^n$

#### 11.5.5 Theorem 7 (Implication of Stationary Source)

Stationary implies Regular

All stationary source are regular. First,  $\lim_{n \rightarrow \infty} H(S_n)$  exists since  $H(S_n) = H(S_1)$  by the definition of stationary source since the index shift does not matter. Then  $\lim_{n \rightarrow \infty} H(S_n | S_1, \dots, S_{n-1})$  exists to since the sequence is non-increasing and bounded below by 0.

#### 11.5.6 Theorem 8 (Relation between Rate and Per symbol Entropies)

For a source  $\mathcal{S} = S_1, \dots, S_n, \dots$

$$H^*(\mathcal{S}) \leq H(\mathcal{S}) \quad (130)$$

#### 11.5.7 Theorem 9 (Cesàro means)

Let  $a_1, \dots, a_n$  be a real value sequence and let  $c_1, \dots, c_n$  be the sequence of running averages defined by

$$c_n = \frac{a_1 + \dots + a_n}{n} \quad (131)$$

If the  $\lim_{n \rightarrow \infty} a_n$  exists, then  $\lim_{n \rightarrow \infty} c_n$  also exists and

$$\lim_{n \rightarrow \infty} c_n = \lim_{n \rightarrow \infty} a_n \quad (132)$$

### 11.5.8 Theorem 10 (Relation with Entropy Rate)

#### Relation with Entropy Rate

A stationary source  $\mathcal{S} = S_1, \dots, S_n, \dots$  is a regular source, furthermore  $\lim_{n \rightarrow \infty} \frac{H(S_1, \dots, S_n)}{n} = H^*(\mathcal{S})$  and  $\frac{H(S_1, \dots, S_n)}{n}$  is non-increasing in  $n$ . We can see that because

$$\frac{H(S_1, \dots, S_n)}{n} = \frac{H(S_1) + H(S_2|S_1) + \dots + H(S_n|S_1, \dots, S_{n-1})}{n} \quad (133)$$

And then, by the theorem of cesàro means, both sides converge to the finite limit  $H(S_n|S_1, \dots, S_{n-1})$  which is  $H^*(\mathcal{S})$

### 11.5.9 Summary of Source Encoding

Let  $\mathcal{S} = S_1, S_2, \dots$  be the infinite sequence produced by a stationary source  $\mathcal{S}$ . By encoding blocks of symbols into D-ary codewords, the average codeword-length per symbol of a uniquely decodable code can be made as close as desired to  $H^*(\mathcal{S})$ . No uniquely decodable D-ary code can achieve a smaller average codeword-length. The above result justifies considering  $H^*(\mathcal{S})$  as a measure of information. In particular,  $H_2^*(\mathcal{S})$  is a measure for the number of bits per source symbol produced by the source  $\mathcal{S}$ .

#### 11.5.10 Independent and Identically Distributed Symbols

For an independent and identically distributed source  $\mathcal{S}$

$$H_D^*(\mathcal{S}) = H_D(\mathcal{S}) \quad (134)$$

## 11.6 Binary Prefix-Free Code for Positive Integers

### 11.6.1 Standard Method

for numbers  $n$

$$l(n) = \lfloor \log_2(n) \rfloor + 1 \quad (135)$$

### 11.6.2 Elias code 1

for numbers  $n$

$$l_1(n) = 2\lfloor \log_2(n) \rfloor + 1 \quad (136)$$

### 11.6.3 Elias code 2

for numbers  $n$

$$l_2(n) = l_1(l(n)) + l(n) - 1 = 2\lfloor \log_2(\lfloor \log_2(n) \rfloor + 1) \rfloor + 1 \quad (137)$$

And for large values

$$l_2(n) \simeq 1 + \log_2(n) \quad (138)$$

## 12 Cryptography

### 12.1 Privacy / Confidentiality

#### 12.1.1 Tools of cryptography

The 3 main tools are

- Authenticate the sender and the receiver.
- Verify the integrity of the message.
- Keep the message confidential.

#### 12.1.2 Description of Cryptography

We have a sender  $A$  (Alice) with a message  $t$  from a sample space  $T$ . The message is encrypted with a key  $E_{k_A}(t)$ . This encryption gives us a cypher-text  $c$  from a sample space  $C$ . The receiver  $B$  (Bob) can have the message with the decoder-key  $D_{k_B}(t)$  where  $D_{k_B}(E_{k_A}(t)) = t$ , the message from the sender.

The security is based on the secret of the key, and not on the secret of the algorithm.

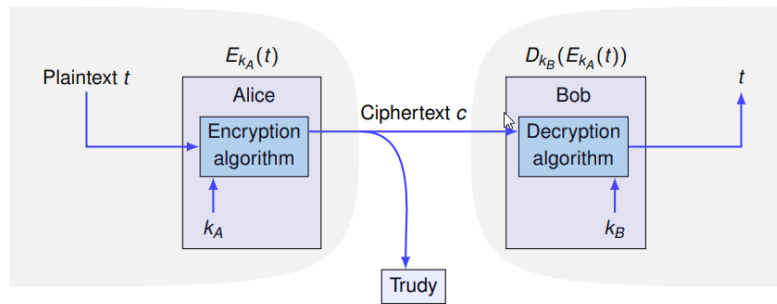


Figure 16: Scheme of Encrypted Communication

#### 12.1.3 Type of Attacks

There are 3 main attacks:

- Cipher-Text-Only: only the cypher-text is known and no other information.
- Known Plain-Text: many plain-texts and cipher-texts are known and all the plain-texts are encrypted with the same key.
- Chosen Plain-Text: For any plain-text that he required, the cryptanalyst can obtain the cypher-text under the same key. It's this attacks that we have to secure ideally.



#### 12.1.4 Definition of Perfect Secrecy

A cryptosystem is perfect secrecy if the plain-text  $t$  and the cypher-text  $c$  are statistically independent. It means that there is no relation between the cypher-text and the plain-text that can give us an advantage on the plain-text knowing the cypher-text only. If we observe the cypher-text, this should not change the attacker's knowledge about the distribution of the plain-text. The attacker can not do better than knowing the plain-text !

Perfect secrecy provides no guarantee against Known-plaintext or Chosen-plaintext attacks. The main weakness is that if we have the plain-text and the cypher-text we can deduce the key  $k = c \oplus t$ . Hence the key should be used only once ! All of this relate that:

$$p(T = t|C = c) = p(T = t) \text{ and } H(T) = H(T|C) \quad (139)$$

#### 12.1.5 Example of Perfect Secrecy: One-Time-Pad (OTP)

For a bits sequence  $t$ , we generate a bits sequence  $d$  as long as  $t$ . To create the cypher-text  $c$ , we just do a modulo 2 between  $t$  and  $d$ . To get the  $t$  back, we have to do  $t = c \oplus d$ . The OTP is perfect secrecy because of the random pad generated. If we know the sequence  $c$ , we can do nothing with this because it's a random bits sequence that encrypted the plain-text before.

#### 12.1.6 Theorem (Perfect Secrecy)

##### Perfect Secrecy

We have the inequality for a plain-text  $T$  and a key  $K$ :

$$H(T) \leq H(k) \quad (140)$$

The proof follows that same idea as the bounds of the entropy.

$$H(T) = H(T|C) \leq H(T, K|C) = H(K|C) + H(T|K, C) = H(K|C) \leq H(K)$$

## 12.2 Public-Key Distribution

### 12.2.1 One-Way Function

It's an algorithm that's easy to compute in one way, but hard in the other.

### 12.2.2 DH-Distribution

Alice takes a number  $a$  and Bob a number  $b$  from a sample space  $A = [1, \dots, p-1]$ . They compute respectively  $g^a$  and  $g^b$  from a generator  $g$  from the sample space  $A$

of numbers. To communicate, they send each other him/his number to compute  $g^{ab}$ . Imagine  $p$  is a 2048-bits number so  $p \cong 2^{2048}$ . To perform  $g^a$  from  $a$ , it takes  $2 \log_2 p = 2 * 2048 = 4096$  multiplications = super easy. Now, to perform  $a$  from  $g^a$ , it takes  $\exp((\frac{64}{9})^{\frac{1}{3}} * (\ln(p))^{\frac{1}{3}} * (\ln(\ln(p)))^{\frac{2}{3}})$  multiplications with  $p = 2^{2048}$  it gives  $10^{35}$  multiplications ! So hard...

### 12.2.3 Paradigm Perfect Secrecy and Security

The perfect secrecy provides full security even if an enemy that has infinite time and resources power cannot break the system. Computational security does not imply the security of the system.

### 12.2.4 Trapdoor One-Way Function

It's a one-way function, but if we know the trapdoor information, then it's easy to decrypt. Now, only Bob has the trapdoor information and so the key can be in the directory. Alice takes the key, encode the message and then only Bob has the trapdoor informations. We define this as an **asymmetric key**.

## 12.3 RSA (Rivest, Shamir, Adleman) Algorithm

### 12.3.1 Construction of the public key

Bob chooses  $p, q$  large and distinct prime numbers. Then he computes  $m = pq$  (it's in  $\mathbb{Z}/m\mathbb{Z}$ ). He chooses an integer  $k$ , multiple of both  $p - 1$  and  $q - 1$ . Appropriate choices:  $k = \text{lcm}(p - 1, q - 1)$  or  $k = \phi(pq) = (p - 1)(q - 1)$ . Then he chooses an exponent  $e \in \mathbb{Z}$  such that  $\gcd(e, k) = 1$ .  $e$  can be the same for all users. The public key is now  $(m, e)$ .

### 12.3.2 Construction of the private key

Since  $\gcd(e, k) = 1 \Rightarrow 1 = ed + kl$  (Bézout theorem), so the private key is  $(m, d)$  and  $[d]_k$  is the multiplicative inverse of  $[e]_k$ . Bob trapdoor one-way function is defined as  $[t]_m \rightarrow [t]_m^e = c$ . Using the private key, we can have the plaintext:  $c^d = ([t]_m^e)^d = [t]_m^{ed} = [t]_m^{1-kl} = [t]_m$  by the CRT and Fermat's theorem.

## 12.4 Privacy for Cryptographic Systems

### 12.4.1 Hash Function

If we have the set of all binary sequences with a many-to-one function  $h(x)$  goes from the last set to the set of all binary sequences of length 200 (so cardinality  $2^{200}$ ), then given  $h(x)$  it is extremely hard to find  $y$  such that  $h(y) = h(x)$ . By definition, a hash function is a one-way-function and it will produce an output specific for a specific input.

### 12.4.2 Digital Signature

How Alice signs a message for Bob ? If  $t$  is the plaintext and  $s = f_A^{-1}(h(t))$  the signature, then  $(t, s)$  is the signed document.  $h(x)$  is the same for everyone. If Alice wants privacy, then she sends  $c = f_B(t, s)$ . To decrypt, Bob compute  $f_B^{-1}(c)$  and then he compares  $h(t) = f_A(s)$ .

### 12.4.3 Trusted Agency

How secure the fact that the key of Bob is the real key and not a key of a villain, all in a public directory ? The directory information is signed by a trusted agency, called S.

- S public key is distributed once and for all by a channel well encoded.
- Each directory entry is digitally signed by S
- Anybody that has S public key can verify that the information received from the directory is authentic.
- Once verified, Alice can be confident.

### 12.4.4 Standarts

Some standarts:

- *SHA1 – SHA3* (Secure Hash Function)
- *DSA* (Digital Signature Algorithm)
- *ECDSA* (Elliptic Curve DSA)
- *DES* (Data Encryption Standart)
- *AES* (Advanced Encryption Standart)
- *RSA* (Rivest, Shamir, Adleman)

### 12.4.5 HTTPS

Hyper-Text Transfer Protocol Secure, used in communication between browser and web server. The browser request something to the web server (service) and the server send back informations with an algorithm from negotiation with a certificate symmetric key.

#### 12.4.6 Example of iMessage (Apple)

When the message service is activated on a device:

- Device produces: RSA keys (1280 bits each) and ECDSA keys (256 bits each).
- IDS (Directory with public keys) stores: user identity (phone email), Device APN (Apple Pushed Notification Service) address, two public keys (RSA, ECDSA).

Alice sends iMessage to Bob:

- The app finds Bob's identifier (Contacts).
- App sends identifier to IDS and gets back: Bob's APN and corresponding keys for each of Bob's device.

For each of Bob's device the following happens:

- A random symmetric key is generated
- The key is encrypted with RSA with a key  $k_A$  and it gives the key  $c_k$ .
- The plaintext is encrypted with AES with a key  $k$  and it gives a ciphertext  $c_t$ .
- The ciphertext and the key are hashed with SHA-1 and then signed with ECDSA, and it gives the state  $S$ .
- At the end, we have the final state  $(c_t, c_k, S)$ .

Bob's device verifies the authenticity of  $c_k$  and  $c_t$  (checking  $S$ ).

- First we decrypt  $c_k \rightarrow k$ .
- Then we decrypt  $c_t$ .

If the plaintext is larger than a certain size:

- A new random key is generated and extra info is AES encoded via the key.
- Cryptogram is uploaded to iCloud.
- (key, URL) are sent as described before.

#### 12.4.7 Complexity of computing $a^k \equiv \text{mod } m$

This complexity is  $2\log_2(m)$  in terms of mod  $m$  multiplications.

#### **12.4.8 Summaty of Symmetric and Asymmetric key Paradigms**

- Symmetric Key: Fast but require to exchange the key over a private channel (DES, AES)
- Asymmetric Key: Slow but no need to exchange the key over a private channel (RSA, El Gamalh).

## 13 Channel Coding

### 13.1 The Noisy Channel

In the transmission of information, after the encryption for example, we have to send it with a physical channel. But before that, a channel encoder (and after a channel decoder) is needed to avoid uncomfortable problems like drops of packets, noisy signals for example.

#### 13.1.1 Erasure Channel

This channel means that some symbols are erased (mark with a question mark symbol here): it's due to drop packets for example. Why ? Because of internet, data storage,... By definition, we say that the erasure weight  $p$  = the number of erasure.

#### 13.1.2 Error Channel

This channel means that some symbols are flipped with another symbols: it's due to electronic noises for example, interferences. Why ? By definition, we say that the error weight  $p$  = the number of errors, for example it's 1 if 0100 becomes 1100.

#### 13.1.3 Detection and Correction

For the error and the erasure channel, we have 2 approaches: the error detection and the error correction. The first is for both erasure and error channel only an asking of retransmission. But for the second, it depends on the channel noise. For the erasure channel, in turns of the question mark symbols, the decoder can get the real value, but it is not always right, for example if the erasure are too many. For the error channel it is not really possible because we have so many possibilities to reconstruct to real input, yes we can say that all possible value of the encoder can be a good value ! We have to find the most similar.

## 13.2 Basics of Channel Coding

### 13.2.1 Terminology

- The code is the set of all codewords:  $C \subseteq A^n$ .
- $n$  is the block length (if  $C = [000, 111]$  then  $n = 3$ ).
- $k$  is the number of information symbols being encoded (if  $00 \rightarrow 000111$  then  $k = 2$  for the 2 '0'). So  $k = \log_{|A|}(|C|)$ .
- $\frac{k}{n}$  is the rate of the code.
- People refer to  $(n, k)$  code !

### 13.2.2 The Hamming Distance

$d(x, y)$  = the number of positions where  $x$  and  $y$  differ. In math, a function of two variables  $x$  and  $y$  is called a distance if it satisfies:

- $d(x, y) \geq 0$  with equality if  $x = y$ .
- Symmetry:  $d(x, y) = d(y, x)$ .
- Triangle:  $d(x, y) \leq d(x, z) + d(z, y)$ .

### 13.2.3 Theorem (Hamming Distance)

Hamming distance is a distance (in sense of math)

### 13.2.4 Hamming-Distance Decoder

Hamming-Distance Decoder

We define  $\hat{c} = \min_{x \in C} d(x, y)$  as the minimizer of  $d(x, y)$  over all  $x \in C$ .

### 13.2.5 The Minimum Sistance of a code C

The minimum (Hamming) distance is the minimum distance between 2 code-words and it's written

$$d_{\min}(C) = \min_{x, x' \in C, x \neq x'} d(x, y) \quad (141)$$

### 13.2.6 What do we want from a Code

- $d_{\min}$  and  $|C|$  should be large (and the rate too).
- $n$  should be small.

### 13.2.7 Singleton Bounds

Regardless to an alphabet  $A$ , the minimum distance of a  $(n, k)$  code satisfies

$$d_{\min} \leq n - k + 1 \quad (142)$$

## 13.3 Minimum Distance (MD) Decoder

### 13.3.1 Theorem (MD-Decoder in Erasure Channel) - Sufficient

If the erasure weight  $p$  is less than  $d_{\min}(C)$  then MD-Decoding is guaranteed to find the current answer. Sufficient because if we have 2 same minimum distance, we can say nothing.

### 13.3.2 Theorem (MD-Decoder in Error Channel)

If the error weight  $p$  is less than  $\frac{d_{\min}(C)}{2}$  then MD-Decoding is guaranteed to find the correct codeword.

–	<i>Erasure channel</i>	<i>Error channel</i>
<i>Detection</i>	<i>not applicable</i>	$p < d_{\min}(C)$
<i>Correction</i>	$p < d_{\min}(C)$	$p < \frac{d_{\min}(C)}{2}$

## 13.4 Code and Linearity

### 13.4.1 Definition of a linear code

A code  $C \subset \mathcal{F}^n$  is linear if  $C$  is a subspace of  $\mathcal{F}^n$ .

- A linear code must contain the all-zero sequence  $\vec{c} = (0, \dots, 0)$ .
- The sum of 2 codewords must be again a codeword.
- The scaled multiplication of a codeword must be again a codeword.

### 13.4.2 Dimension of a Linear Code

#### Dimension of a Linear Code

The dimension of a linear code  $C$  is the dimension of the subspace  $C$ . The number of codewords in a linear code  $C \subset \mathcal{F}^n$  must be of the form  $q^k$  where  $k \in [1, \dots, n]$  and where  $q$  must be the form  $q = p^m$  with  $p$  prime and  $m \in \mathbb{Z}_+$ . For example  $C \subset \mathcal{F}_2^7$  and  $|C| = 8 = q^k = 2^3$  because  $q$  must be the size of the field.

## 13.5 Hamming Weight

### 13.5.1 Definition

Let  $\vec{x} \in \mathcal{F}^n$ . The hamming weight of  $\vec{x}$  is  $w(\vec{x}) = d(\vec{x}, \vec{0})$  where  $d$  is the hamming distance.

So the hamming weight is the number of non-zero positions in  $\vec{x}$ .

### 13.5.2 Theorem of Minimum Distance

Let  $C \subset \mathcal{F}^n$  be a linear code. Then  $d_{\min}(C) = \min_{\vec{x} \in C, \vec{x} \neq \vec{0}} w(\vec{x})$ .



## 13.6 Basis and Description of Code

A linear code can be described by a basis  $[\vec{c}_1, \dots, \vec{c}_k]$  where  $k$  is the dimension of the code. The matrix formed is called the generator matrix of the code written  $G$ . The matrix specifies an encoding map  $f$ .

$$f : \mathcal{F}^k \rightarrow \mathcal{F}^n \quad \vec{u} \rightarrow \vec{u}G = \vec{c}$$

The generator is not unique because the basis too. The number  $n$  of generator matrix  $[\vec{c}_1, \vec{c}_2, \dots, \vec{c}_k]$  is computed by:

$$n = (q^k - 1)(q^k - q)(q^k - q^2) \dots (q^k - q^{k-1}) \quad (143)$$

and more generally

$$n = \prod_{i=0}^{k-1} (q^k - q^i) \quad (144)$$

### 13.6.1 Systematic Generator Matrices

A systematic generator matrix is a generator matrix of the form

$$G = \begin{pmatrix} 1 & 0 & 0 & a_1 & \dots & a_n \\ 0 & 1 & 0 & b_1 & \dots & b_n \\ 0 & 0 & 1 & c_1 & \dots & c_n \end{pmatrix}$$

Where the lines must create a identity matrix at beginning and then arbitrary coefficients. This identity matrix provides an independence between the lines.

Warning, this matrix does not always exist !

## 13.7 Decoding

Decoding is about deciding the information word from the output channel. If the output information is a codeword, we assume that it's equal to the channel input. In the case decoding is about inverting the encoding map. But how to we determine efficiently if the channel output is a codeword ? We use the fact that in linear code, like every subspace of a vector space, can be defined by a system of linear homogeneous equations. The channel output is a codeword if and only if it satisfies those equations.

### 13.7.1 Error Pattern

By definition, we can always represent an error writing an information  $\vec{y} = \vec{x} + \vec{e}$  where  $\vec{e}$  is the error applied to  $\vec{x}$  that gives the error information of  $\vec{y}$ .

### 13.7.2 Parity Check Matrix

A parity check matrix  $H$  for a linear  $(n, k)$  code  $C$  is an  $((n - k) \times n)$  matrix that contains the coefficient of  $n - k$  linear homogeneous equations the solutions to which are  $C$ .

This matrix has this properties extremely important:

$$\vec{y}H^T = \vec{0} \Leftrightarrow \vec{y} \in C \text{ and } GH^T = \vec{0}$$

To compute the parity check matrix from a generator matrix, we can do:

$$G = (I_k | P) \Leftrightarrow H = (-P^T | I_{n-k})$$

### 13.7.3 Syndrome

#### Syndrome of an element

The vector  $\vec{s} = \vec{y}H^T \in \mathcal{F}^{n-k}$  is the syndrome of  $\vec{y}$ . By definition of the parity check matrix, now:

$$\vec{s} = \vec{0} \Leftrightarrow \vec{y} \in C \quad (145)$$

### 13.7.4 Theorem Minimum Distance

The minimum distance of a code is the smallest number of linear dependent columns of the parity matrix  $H$ .

### 13.7.5 Dimension of a Code with Parity Matrix

As we know, the dimension of a subspace  $S$  of a vector space  $V$  can be found with the fact that the dimension is equal to  $n - m$  with  $n$  is the dimension of the space vector  $V$  and  $m$  are the number of linear homogeneous equations that spans the code. So  $m$  is the number of rows of the parity matrix and  $n$  the number of columns.

## 13.8 Decoding based on Cosets and Syndromes

### 13.8.1 Equivalence Relations with Codes

We see in the group theory part that when  $\mathcal{G}$  forms a commutative group  $(\mathcal{G}, +)$  and  $(\mathcal{H}, *)$  is a subgroup of  $\mathcal{G}$ , there is a natural choice for  $\sim$  defined as follows:

$$a \sim b \text{ if there exists an } h \in \mathcal{H} \text{ such that } b = a * h \quad (146)$$

We arrive then at the next definition

### 13.8.2 Cosets

If  $(G, +)$  is a group and  $(H, +)$  a subgroup of  $G$ , then write  $a \sim b$  if  $b = a + h$  for some  $h \in H$  and  $a, b \in G$ . The equivalence class of  $a \in G$ :

$$[a] = [b : a \sim b] = [a + h, h \in H] = a + H \quad (147)$$

And we say that it's the coset of  $H$  with respect of  $a$ . For example if  $G = (Z/10Z, +)$  and  $H = \{0, 5\}$ , then  $[0] = [0+0, 0+5] = \{0, 5\}$ ,  $[1] = [1+0, 1+5] = \{1, 6\}$ , ... and so on for every element in  $G$ .

### 13.8.3 Facts on cosets

- Every  $b \in G$  is in exactly one coset.
- All cosets have the same cardinality
- Cosets are either equal or full distinct
- The union of all cosets gives  $G$ .

### 13.8.4 Application of Cosets on Code

Now for us the group  $G$  is the the finite field  $(\mathcal{F}^n, +)$  and the subgroup  $H$  is the code  $C$ . Then

- If  $x, y \in \mathcal{F}^n$ ,  $x \sim y \Leftrightarrow -x + y \in C$
- Equivalently,  $x \sim y \Leftrightarrow y = x + c$  for some  $c \in C$

### 13.8.5 Standard Array

With the figure, we see that the codewords of the code are in the top row. And each rows below are the cosets of the the code with respect of a codeword. Each element in the array are not already in a row because if it was, the coset will be the same! So in each row, we add all element of  $C$  to a element coming from the vector space  $\mathcal{F}$ .  $L = q^{n-k}$  because if  $M = q^k$  we have to get  $q^n$  with  $q^k q^{n-k}$ . Each column of the coset are denoted by  $D_j$ . We have at the end in the array every element of  $\mathcal{F}$  once.

### 13.8.6 Coset Decoder

Now that we have a standard array, we could say that when we have an output codeword, then we search it in the array and we took the codeword at the top of the  $D_j$  row. But in practical, this is not worse and too complicated. What we want is to define **coset leaders** for each row. Let's define an error probability pattern such that a bit has probability  $\epsilon$  so switch and probability  $1 - \epsilon$  to not switch, then we came to the error pattern of  $e \in \mathcal{F}^n$

$$\epsilon^w(e)(1 - \epsilon)^{n-w(e)} \quad (148)$$

$c_0 = 0$	$c_1$	$c_2$	$\dots$	$c_{M-1}$	$\leftarrow [C]$
$t_1$	$t_1 + c_1$	$t_1 + c_2$	$\dots$	$t_1 + c_{M-1}$	$\leftarrow [t_1]$
$t_2$	$t_2 + c_1$	$t_2 + c_2$	$\dots$	$t_2 + c_{M-1}$	$\leftarrow [t_2]$
$\vdots$	$\vdots$	$\vdots$		$\vdots$	$\vdots$
$t_{L-1}$	$t_{L-1} + c_1$	$t_{L-1} + c_2$	$\dots$	$t_{L-1} + c_{M-1}$	$\leftarrow [t_{L-1}]$

where for each  $j = 1, \dots, L-1$ ,  $t_j$  is such that

$$t_j \notin \left( C \bigcup_{k=1}^{j-1} [t_k] \right).$$

Figure 17: Standard Array Of elements of a Code

After that, we can derive this to the fact that the error probability is minimized when the coset leaders have the smallest weight. We conclude then with that  $D_0$  is the row with the cosets leaders and  $D_i = D_0 + c_i$ .

### 13.8.7 Theorem (Relation between Cosets and Syndromes)

#### Cosets and Syndromes

Let  $\vec{y}_1$  and  $\vec{y}_2$  be vectors in  $\mathcal{F}^n$ . Then:

$\vec{y}_1, \vec{y}_2$  have the same syndrome  $\Leftrightarrow$  they are in the same coset.

By this, we can say that the syndrome uniquely identify the coset leader !

### 13.8.8 Decoding Algorithm Explanation

With this standard array and cosets leader, we pre-compute the cosets leader and their syndromes. To decode  $y$  the channel output, we compute the syndrome  $s = yH^t$  of  $y$ , so  $s$  encodes the row of  $y$ . Then we identify the coset leader with the syndromes, say  $t_i$ .  $t_i$  and  $y$  uniquely determine the column of  $y$ , so  $y = t_i + c_j$ . Hence  $c_j = y - t_i$  and so the decoder declares that the codeword from the input channel is  $\hat{c} = y - t_i$ .

Then to find the information word  $\hat{u}$ , we compute  $\hat{u}G = \hat{c}$ .

<i>Coset leader</i> $\vec{t}_i$	<i>Syndrome</i> $\vec{t}_i H^T$
$\vec{t}_0 = \vec{0}$	$\vec{s}_0 = \vec{0}$
$\vec{t}_1$	$\vec{s}_1 = \vec{t}_1 H^T$
$\dots$	$\dots$
$\vec{t}_{L-1}$	$\vec{s}_{L-1} = \vec{t}_{L-1} H^T$

### 13.8.9 Examples of Coset Decoding

Let  $C = \{c_1 = (0, 0, 0), c_2 = (1, 1, 1)\}$ . Then let the  $t_i$  be element of  $\mathcal{F}_2^3$ :  $(1, 0, 0), (0, 1, 0), (0, 0, 1)$ . We compute the array:

$c_0 = (0, 0, 0)$	$c_1 = (1, 1, 0)$	$\leftarrow C$
$t_1 = (0, 0, 1)$	$t_1 + c_1 = (1, 1, 1)$	$\leftarrow t_1 + C$
$t_2 = (0, 1, 0)$	$t_2 + c_1 = (1, 0, 1)$	$\leftarrow t_2 + C$
$t_3 = (1, 0, 0)$	$t_3 + c_1 = (0, 1, 1)$	$\leftarrow t_3 + C$

The first column is  $D_0$  and the second  $D_1 = D_0 + c_1$ . Let say we have a parity matrix  $H$ :

$$H^T = \begin{pmatrix} 1 & 1 \\ 0 & 1 \\ 1 & 0 \end{pmatrix} \quad (149)$$

and the coset-syndromes array

$t$	$s$
$(0, 0, 0)$	$(0, 0)$
$(0, 0, 1)$	$(0, 1)$
$(0, 1, 0)$	$(1, 0)$
$(1, 0, 0)$	$(1, 0)$

If  $y = (1, 0, 1)$  is received, then the syndrome  $s = yH^T = (0, 1)$ , the coset leader of  $(1, 0)$  is  $(0, 1, 0)$  and so the decoded codeword is  $\hat{c} = y - t = (1, 1, 1)$ .

## 13.9 Reed-Solomon Codes (RS)

### 13.9.1 Lagrange's Interpolation

The problem is: given a field  $\mathcal{F}$  and  $k$  pairs  $(a_i, y_i) \in \mathcal{F}^2$  where the  $a_i$  are all distinct, is there a polynomial  $P(x)$  over the field of degree at most  $k - 1$  such that  $P(a_i) = y_i$  for all  $i$ ? Yes and this is how it works:

Fix elements  $(a_1, y_1), (a_2, y_2), (a_3, y_3)$  and so we seek a polynomial  $P(x)$  of degree at most 2 and coefficients in the field such that  $P(a_i) = y_i$ . Suppose we can find a polynomial  $Q_1(x)$  of degree at most 2 such that

$$Q_1(x) = \begin{cases} 1 & x = a_1 \\ 0 & x = a_i \neq a_1 \end{cases} \quad (150)$$

and similarly for  $Q_2(x)$  and  $Q_3(x)$ . The  $P(x)$  is then  $y_1Q_1(x) + y_2Q_2(x) + y_3Q_3(x)$ . The construction for  $Q_1(x)$  is

$$Q_1(x) = \frac{(x - a_2)(x - a_3)}{(a_1 - a_2)(a_1 - a_3)} \quad (151)$$

and similarly for the others. More generally, it's denoted by this equation

$$P(x) = \sum_{j=0}^{d-1} b_j \prod_{k \neq j} \frac{x - a_k}{a_j - a_k} \quad (152)$$

### 13.9.2 Fundamental Theorem of Algebra

#### Fundamental Theorem of Algebra

Let  $P(x)$  a polynomial of degree at most  $k$  over a field. If  $P(x) \neq 0$  then the number of its distinct roots is at most  $k$ .

### 13.9.3 Construction

Choose a finite field  $\mathcal{F}$  and integers  $k, n$  such that  $0 < k \leq n \leq q$  where  $q = |\mathcal{F}|$ . Then choose  $n$  distinct elements  $a_1, \dots, a_n \in \mathcal{F}$ . The codewords are defined via the map

$$\mathcal{F}^k \rightarrow \mathcal{F}^n \quad \vec{u} \mapsto \vec{c} = (P_u(a_1), \dots, P_u(a_n)) \quad (153)$$

Where  $P_u(a_i)$  is a polynomial over the finite field evaluated on  $a_i$ . The elements  $a_i$  are the different symbols that can be sent and the polynomial is what gives us the codebook. And it's by the Lagrange's interpolation that we can find a polynomial like this, that can code every word that we want with the same polynomial.

### 13.9.4 Properties of RS Codes

- They are MDS and Linear
- Nice construction and an efficient decoding algorithm.

### 13.9.5 Example of RS Code

Let fix the elements in  $\mathcal{F}_3^2$

$u$	$P_u(x)$	$c$
00	0	000
01	$x$	012
02	$2x$	021
10	1	111
11	$1 + x$	120
12	$1 + 2x$	102
20	2	222
21	$2 + x$	201
22	$2 + 2x$	210

The parity matrix must be the dimension  $k = n - m$  so  $3 - 2 = 1$ , so 1 row. We see that for 10, we encode to 111, so  $(1, 0)G = (1, 1, 1)$  so the first row of  $G$  is  $(1, 1, 1)$ . For  $(0, 1)G = (0, 1, 2)$ , so the second row is  $(0, 1, 2)$ , then  $G$  is the 2 lines here.