Chapter 1

Preliminaries

In this chapter we compile the basic concepts of complexity theory, propositional logic, and probability theory which will be continuously used in this work. Complexity theory is needed to evaluate the efficiency of the introduced methods that allow for an analysis of interaction networks, in particular, it enables us to compare it with other concepts. Propositional logic is fundamental to all models for interaction networks that are developed herein. Finally, the theory of probability is needed in Chapter 4, where we analyze a randomized algorithm. Throughout this work we use the theory of polyhedra, integer programming, and combinatorial optimization hand in hand with propositional logic. We assume that these concepts are known and refer to the books [BW05], [Sch86], and [Sch03a] for details.

1.1 Complexity Theory

Complexity theory is a tool to characterize the degree of difficulty of a *problem*. In this context a problem is a general question to be answered with several parameters. An *instance* of a problem is obtained by specifying the parameters. An *algorithm* is a procedure to solve an instance of the problem. This section will give a brief introduction to this topic. The definitions are according to [GJ79].

Running time The running time of an algorithm measures the space and time used to solve a class of problems with the help of a certain algorithm. Intuitively, the performance of the algorithm depends on the size of its input. Thus, every instance of a problem class is assigned an *encoding length* or *input size* which specifies the size of the input data given to the algorithm. The time and space complexity of the algorithm is then determined in relation to this input size.

In the usual encoding scheme the encoding length of a problem is measured by the number of bits necessary to save the data. For an integer n we exploit the standard binary encoding leading to an input size of $\langle n \rangle = \lceil \log_2(|n|) + 1 \rceil + 1$, where the latter 1 refers to the encoding size of the sign of n. Hence, the encoding length of a rational number r given as p/q, with p and q integers, is $\langle r \rangle = \langle p \rangle + \langle q \rangle$. Analogously, the input size of a matrix is the sum of the sizes of its entries.

To measure the complexity of an algorithm the growth of the time (space) requirement is analyzed with respect to the growth in input size. For this the unit operations – addition, subtraction, multiplication, division and comparison – are counted and their unit times are accumulated. This leads to the *time complexity function* $f : \mathbf{N} \to \mathbf{N}$ of an algorithm defined as the largest amount of time $f(n)$ needed for the algorithm to solve a problem instance of each possible input size $n \in \mathbf{N}$. Similarly, the *space complexity function* $g : \mathbf{N} \to \mathbf{N}$ describes the maximum space $g(n)$ needed to solve an instance of an encoding length of at most $n \in \mathbf{N}$. The space complexity of an algorithm can be incorporated into the time complexity by counting the writing on and reading of the memory as unit operations, which we will do from now on. In this setting the space complexity is dominated by the time complexity so that we can focus on the time complexity function.

Since one is usually interested in the order of magnitude of the running time of an algorithm rather than in the exact time complexity function, we use the asymptotic Landau notation to describe the growth of a function. For two functions $f, h : \mathbf{N} \to \mathbf{R}$ we say that $f(n) \in O(h(n))$ if there exist positive constants $c \in \mathbf{R}$ and $n_0 \in \mathbf{N}$ such that

$$\left|f(n)\right| \leq c \left|h(n)\right| \quad \text{for all } n \geq n_0.$$

To indicate that the constant $c = c(\epsilon)$ depends on a constant $\epsilon \in \mathbf{R}$, we denote this by $f(n) \in O_\epsilon(h(n))$. Further, $f(n) \in \Theta(h(n))$ if

$$f(n) \in O(h(n)) \quad \text{and} \quad h(n) \in O(f(n)).$$

Finally, we have $f(n) \in o(h(n))$ if there exists $n_0 \in \mathbf{N}$ such that *for all* positive $c \in \mathbf{R}$,

$$\left|f(n)\right| \leq c\left|h(n)\right| \quad \text{for all } n \geq n_0.$$

The $o(h(n))$ indicates actual difference in the order of magnitude of f and h, like e.g., $f(n) = \log n$ and $h(n) = n$ while $\Theta(h(n))$ refers to the same size of $f(n)$ and $h(n)$, as e.g., $f(n) = 5n^2 + n$ and $h(n) = n^2$.

The Landau symbols allow for the characterization of "easier" and "harder" problems. Each algorithm with a time complexity function $f(n) \in O(p(n))$ for a polynomial p is called a polynomial time algorithm. An algorithm that is polynomial time in the *sum* of the sizes of its input and output is called *output polynomial*. If p is a linear function, we speak of a linear time algorithm while for p being an exponential function we have an exponential time algorithm. A running time that lies between polynomial and exponential time is the *quasi-polynomial* time, that is, if the time complexity function $f(n)$ of an algorithm is in $O(2^{\text{polylog}(n)})$.

$\mathcal{P}$ versus $\mathcal{NP}$ In this paragraph the problems are restricted to so called *decision problems*. The only possible answers to these problems are '`yes`' or '`no`'. One is interested in categorizing decision problems into different classes.

The class $\mathcal{P}$ contains all decision problems for which there exists an algorithm with time complexity function f, $f \in O(p)$ for a polynomial p, that solves it.

The class $\mathcal{NP}$ includes all decision problems whose '`yes`' answer can be verified in polynomial time. $\mathcal{NP}$ is short for the class of nondeterministic polynomial decision problems. It is certainly clear that $\mathcal{P} \subseteq \mathcal{NP}$, but it is an open question whether $\mathcal{P} = \mathcal{NP}$.

Example 1.1. Let $\mathbf{c} \in \mathbf{Z}^n, \mathbf{A} \in \mathbf{Z}^{m\times n}$, and $\mathbf{b} \in \mathbf{Z}^m$. Consider the binary integer optimization problem

$$\begin{aligned} \max \quad & \mathbf{c}^\intercal \mathbf{x} \\ \text{s. t.} \quad & \mathbf{x} \in \mathcal{F} = \{\mathbf{x} \in \{0,1\}^n \mid \mathbf{A}\mathbf{x} \leq \mathbf{b}\}. \end{aligned}$$

The corresponding decision problem is the question whether there is an $\mathbf{x} \in \mathcal{F}$ so that $\mathbf{c}^\intercal \mathbf{x} \geq \alpha$ for a given constant $\alpha \in \mathbf{Z}$. It is in the class $\mathcal{NP}$

as a solution $\mathbf{x}_0 \in \{0,1\}^n$ is of polynomial size and it can be verified by testing whether $\mathbf{c}^\intercal \mathbf{x}_0 \geq \alpha$, using n multiplications, $n-1$ additions, and 1 comparison. To check if $\mathbf{x}_0 \in \mathcal{F}$ takes $2nm$ operations. Altogether, $(m+1)2n$ operations are necessary to confirm a 'yes' answer, thus the verification needs $O(m \cdot n)$ steps.

Another complexity class closely related to $\mathcal{NP}$ is the class co-$\mathcal{NP}$. It contains all decision problems whose 'no' answer can be verified in polynomial time, or alternatively it contains all problems whose complement is in $\mathcal{NP}$. An example is the question whether a given number $k \in \mathbf{N}$ is prime. If it is not prime, this can be verified in polynomial time with an integer $d \in \mathbf{N}$ that divides k. It is not known whether $\mathcal{NP}$ = co-$\mathcal{NP}$.

A key question is the comparability of two decision problems – when are two decision problems equally hard? Suppose two decision problems Π_1 and Π_2 are given. A *polynomial transformation* is an algorithm transferring an instance σ of Π_1 in polynomial time to an instance σ' of Π_2 such that the answer to σ is 'yes' if and only if the answer to σ' is 'yes'. A decision problem Π is called $\mathcal{NP}$*-complete* if two conditions hold: $\Pi \in \mathcal{NP}$, and an $\mathcal{NP}$*-complete* problem can be polynomially transformed to Π. In particular, this implies that if one $\mathcal{NP}$*-complete* problem is polynomially solvable, all $\mathcal{NP}$*-complete* problems are in $\mathcal{P}$. A polynomial algorithm for an $\mathcal{NP}$*-complete* problem further implies $\mathcal{P} = \mathcal{NP}$. Nevertheless, there can exist polynomial solution procedures for particular subclasses of problems in $\mathcal{NP}$ or co-$\mathcal{NP}$. These procedures can usually be obtained by exploiting special combinatorial features or structures of the subclasses. This closes our brief overview of complexity theory. For a detailed introduction see [GJ79].

1.2 Propositional Logic

For this short summary we follow the notation of Kleine Büning and Lettman [KBL99].

Definition 1.2 (atom)**.** An *atom* is a basic proposition which can only possess two values, True or False.

Atoms can be combined by logical 'and' (denoted by $\wedge$), 'or' (denoted by $\vee$), and 'not' (denoted by $\neg$) operations to logical formulas. These operations are called conjunction, disjunction, and negation, respectively.

Remark 1.3. If it is not to be emphasized whether an atom is negated or not, an atom or its negation is called *literal*. Sometimes an atom is referred to as positive literal and a negated atom as negative literal.

Definition 1.4 (propositional formula)**.** The set of *propositional formulas* is inductively defined by four rules:

(i) Every atom is a formula.

(ii) If α is a formula, then $(\neg\alpha)$ is also a formula.

(iii) If α and β are formulas, then $\alpha \vee \beta$ and $\alpha \wedge \beta$ are also formulas.

(iv) Only expressions according to (i)-(iii) are formulas.

Two special logical operations we continuously use are the *implication* '$\rightarrow$' and the *equivalence* '$\leftrightarrow$'. Both are abbreviations for combinations of standard operations. Let α and β be two propositional formulas, then $\alpha \rightarrow \beta$ denotes $\neg\alpha \vee \beta$ and $\alpha \leftrightarrow \beta$ means $(\alpha \rightarrow \beta) \wedge (\beta \rightarrow \alpha) \approx (\neg\alpha \vee \beta) \wedge (\neg\beta \vee \alpha)$, where '$\approx$' denotes logical equivalence.

There are different standard forms of propositional formulas with distinct properties. The most commonly used is the *Conjunctive Normal Form*.

Definition 1.5 (clause, Conjunctive Normal Form, Horn clause, Horn formula)**.**

1. A *clause* α is a disjunction of literals, i.e., $\alpha = A_1 \vee \ldots \vee A_n$ with literals A_i. It is called k-clause if it contains at most k literals. A clause is called *empty* if it contains no literal.

2. A formula α is in *Conjunctive Normal Form* (CNF) if and only if α is a conjunction of clauses.

3. A formula α is in k-CNF if and only if α is a conjunction of k-clauses.

4. A clause is called a *Horn clause* if it contains at most one positive literal.

5. The conjunction of Horn clauses is named a *Horn formula*.

Note that each propositional formula can be transformed into a logically equivalent formula in k-CNF, $k \geq 3$, in polynomial time.

An important question in propositional logic is whether there exists a truth assignment for literals, i.e., an assignment of 0 or 1 to each atom, such that a propositional formula is True.

Definition 1.6 (truth assignment, satisfiable)**.**

1. A *truth assignment* Γ of a propositional formula α is defined as

$$\Gamma : \{\alpha \mid \alpha \text{ is a propositional formula}\} \to \{0, 1\}.$$

   For propositional formulas α and β it is calculated according to three rules:

   (i) $\Gamma(\neg\alpha) := 1$ if and only if $\Gamma(\alpha) = 0$;

   (ii) $\Gamma(\alpha \vee \beta) := 1$ if and only if $\Gamma(\alpha) = 1$ **or** $\Gamma(\beta) = 1$;

   (iii) $\Gamma(\alpha \wedge \beta) := 1$ if and only if $\Gamma(\alpha) = 1$ **and** $\Gamma(\beta) = 1$.

2. A propositional formula α is *satisfiable* if and only if there exists a truth assignment Γ so that $\Gamma(\alpha) = 1$.

Note that an empty clause is, by definition, not satisfiable. The set of satisfiable formulas in CNF is denoted by

$$\text{SAT} := \{\alpha \in \text{CNF} \mid \alpha \text{ is satisfiable}\}$$
$$k\text{-SAT} := \{\alpha \in k\text{-CNF} \mid \alpha \text{ is satisfiable}\}.$$

Since every propositional formula can be transformed into a logically equivalent formula in 3-CNF in polynomial time, testing an arbitrary propositional formula for satisfiability can be reduced to testing its equivalent 3-CNF. But this still remains hard:

Theorem 1.7 ([Coo71])**.** *SAT and* 3*-SAT are* $\mathcal{NP}$*-complete.*

In this context Horn as well as 2-CNF formulas play an important role because testing them for satisfiability can be accomplished in polynomial time. Horn formulas can be tested for satisfiability by means of value propagation using the logical programming language Prolog [Rou75]. For 2-CNF formulas there exists a graphical infeasibility certificate of linear size which was introduced by Aspvall, Plass, and Tarjan [APT79]. They reduce unsatisfiability of a 2-CNF instance α_2 to a property of a certain auxiliary digraph $G(\alpha_2)$ that is constructed according to three rules:

(i) Introduce the two nodes A and $\neg A$ for every atom A in α_2.

(ii) For every 2-clause $A \vee B$ the two arcs $(\neg A, B)$ and $(\neg B, A)$ are introduced.

(iii) For every 1-clause A make the arc $(\neg A, A)$.

The arcs can be seen as logical consequences. Whenever the tail of an arc evaluates to TRUE, the head also has to evaluate to TRUE in order to satisfy the associated 2-clause. The following certificate is obtained from this construction.

Theorem 1.8 ([APT79]). *A 2-CNF formula α_2 is satisfiable if and only if no pair of vertices A and $\neg A$ occurs in one strongly connected component of $G(\alpha_2)$.*

This means that α_2 is not satisfiable if there exists a path in $G(\alpha_2)$ leading from A to $\neg A$ together with a path from $\neg A$ to A. The one path states that A must be FALSE while the other requires the opposite, a contradiction.

In case α_2 is satisfiable, the satisfying assignment is obtained by traversing the strongly connected components of $G(\alpha_2)$ in reverse topological order, computed by the linear time algorithm of Tarjan [Tar72], while obeying that every pair of vertices A and $\neg A$ are assigned complementary values and for no arc (A, B) the head B is assigned FALSE and the tail A is TRUE.

A purely syntactical method to test the satisfiability of a CNF formula is *resolution* which is a recursive elimination of the variables. The only rule of the resolution calculus is that for two clauses $(\alpha \vee A)$ and $(\beta \vee \neg A)$ the new clause $(\alpha \vee \beta)$ can be concluded. An empty clause is produced in the resolution procedure precisely if the original CNF is unsatisfiable. Just as the Fourier-Motzkin Elimination [Mot36] the resolution procedure can yield a quadratic increase in the size of the CNF formula in every step.

One last notion of a propositional formula is needed (see e.g., [KSS00]).

Definition 1.9 (model, $\leq$, maximal (minimal) model, monotone)**.**

1. A *model* m of a propositional formula α is a satisfying truth assignment of α. The set of all models of α is denoted by models(α).

2. Let Γ_1 and Γ_2 be two truth assignments of α. If for all atoms A of α for which $\Gamma_1(A) = 1$ it also holds that $\Gamma_2(A) = 1$, then $\Gamma_1 \leq \Gamma_2$.

3. A model m^* is called *maximal (minimal)* if no other model m exists such that $m^* < m$ ($m^* > m$). The set of all maximal (minimal) models of α is denoted by maximal(α) (minimal(α)).

4. A formula α is called *up-monotone* if all truth assignments $\Gamma \geq m$, for a model m, are also models of α, and it is named *down-monotone* if

all $\Gamma \leq m$, for a model m, are also models of α. If α is either up- or down-monotone, it is *monotone*.

An example of an up-monotone formula is a CNF formula with only positive literals. Equivalently, a CNF with only negative literals is down-monotone. Clearly, the question of satisfiability for monotone formulas is therefore easy: If **1** or **0**, respectively, satisfies the formula, it is satisfiable. In this context it is rather of interest to identify the irredundant dual of a monotone CNF formula. This can be computed in *incremental* quasi-polynomial time using the *Joint Generation algorithm* [GK99], that is, in every step the algorithm generates new output in quasi-polynomial time. Crucial to this algorithm is the result of Fredman and Khachiyan [FK96] which ensures that deciding the duality of two monotone propositional formulas can be done in $n^{o(\log n)}$, where n is the size of the two monotone formulas. For an overview of the Joint Generation method see [EMG08]. In particular, the Joint Generation algorithm is a smart, general purpose enumeration method for monotone generation problems. It simply requires an oracle that monotonously decides for feasibility. This makes it a powerful tool for theory and application. In the subsequent chapters we will present some of the numerous applications.

1.3 Probability Theory

For the introduction to basic probability theory we follow, if not indicated otherwise, the book [MU05]. All proofs are left out and can be found in the book. Initially, we need to define a *probability space* which underlies every probabilistic analysis.

Definition 1.10 (probability space, probability function)**.** A probability space is a 3-tuple $(\Omega, \mathcal{E}, \mathbf{P})$ containing the following three components:

1. the *sample space* Ω consisting of all possible outcomes of the random experiment,

2. a family of sets $\mathcal{E}$, where each $E \in \mathcal{E}$ is a subset of Ω, which represents all allowable events, and

3. a *probability function* $\mathbf{P} : \mathcal{E} \to [0, 1]$ satisfying the subsequent conditions:

   (i) $\mathbf{P}(\emptyset) = 0$, $\mathbf{P}(\Omega) = 1$, and

(ii) for any finite or countably infinite sequence of pairwise disjoint sets $E_1, E_2, \ldots$ it holds true that

$$\mathbf{P}\left(\bigcup_{i\geq 1} E_i\right) = \sum_{i\geq 1} \mathbf{P}(E_i). \tag{1.1}$$

For discrete probability spaces the sample space Ω is finite or countably infinite, and the family $\mathcal{E}$ of allowable events consists of subsets of Ω. To illustrate the definition consider the example of throwing a die. Then $\Omega = \{1, 2, \ldots, 6\}$, $\mathcal{E}$ consists of each subset with one element, and $\mathbf{P}(i) = \frac{1}{6}$ for all $1 \leq i \leq 6$.

From now on we always assume a probability space $(\Omega, \mathcal{E}, \mathbf{P})$, and denote the complement of an event E by $\overline{E} = \Omega \setminus E$. For every two events $E_1, E_2 \in \mathcal{E}$ the following useful property is easily concluded

$$\mathbf{P}(E_1 \cup E_2) = \mathbf{P}(E_1) + \mathbf{P}(E_2) - \mathbf{P}(E_1 \cap E_2) \leq \mathbf{P}(E_1) + \mathbf{P}(E_2),$$

where we use the usual notation of set theory. A useful version for n events $E_1, \ldots, E_n$ is the inequality

$$\mathbf{P}\left(\bigcup_{i=1}^{n} E_i\right) \leq \sum_{i=1}^{n} \mathbf{P}(E_i).$$

Note that the difference to Equation (1.1) is that in this case the events must not be pairwise disjoint. For the intersection of two events a different property of the E_i ensures such kind of equality. The events $E_1, \ldots, E_n$ are mutually *independent* if for all subsets $L \subseteq \{1, \ldots, n\}$

$$\mathbf{P}\left(\bigcap_{i\in L} E_i\right) = \prod_{i\in L} \mathbf{P}(E_i).$$

If we throw another die together with our first one, the two events that each die shows a '1' are independent. Hence, the event that both show a '1' has a probability of $\frac{1}{36}$.

Often one aims at analyzing the probability of an event E under the condition that a certain other event F takes place. For example, we might want to compute the probability that it snows provided that it is summer. This probability is certainly smaller than if we assume winter time.

Definition 1.11 (conditional probability)**.** The *conditional probability* of event E given that event F occurs is

$$\mathbf{P}(E \mid F) := \frac{\mathbf{P}(E \cap F)}{\mathbf{P}(F)}.$$

It is only well-defined if $\mathbf{P}(F) > 0$.

The conditional probability can be seen as looking at the event $E \cap F$ among those sets containing F. An immediate, easy consequence is that $\mathbf{P}(E \mid F) = \mathbf{P}(E)$ if and only if E and F are independent. Applying the conditional probability with a partition $E_1, \ldots, E_n$ of Ω yields a useful instrument to determine the probability of an event.

Theorem 1.12 (Law of Total Probability)**.** *Let F be an event and $E_1, \ldots, E_n$ a partition of Ω, then*

$$\mathbf{P}(F) = \sum_{i=1}^{n} \mathbf{P}(F \cap E_i) = \sum_{i=1}^{n} \mathbf{P}(F \mid E_i)\, \mathbf{P}(E_i).$$

Random variables When studying random events one is often interested in an associated value rather than in the event itself. One can be interested in the sum of the results of two dice, and with which probability the sum equals a given value. In order to do so *random variables* are introduced. As we will mostly need discrete random variables, we focus on these. The given definitions remain valid for continuous random variables if we assume a continuous probability function and use appropriate integrals instead of sums.

Definition 1.13 ((discrete) random variable)**.** A *random variable* X on a sample space Ω is a real-valued function $X : \Omega \to \mathbf{R}$. A *discrete* random variable is a random variable that attains a finite or countably infinite number of values.

Random variables are always denoted by capital letters while real numbers are presented as lowercase letters. For a discrete random variable X and a real number a the event '$X = a$' includes all elements of Ω for which the random variable is equal to a. The probability of this event therefore is

$$\mathbf{P}(X = a) = \sum_{s \in \Omega : X(s) = a} \mathbf{P}(s).$$

With this, the probability $\mathbf{P}(X = a)$ defines a function $f : \mathbf{R} \to [0, 1]$ as $f(a) = \mathbf{P}(X = a)$, called the *probability mass function* of X. Let us consider the two dice we throw, and assume we are interested in the probability that the sum of the two dice is 4. We define X as the sum of results of the two dice thrown once. For each (i, j), $1 \leq i, j \leq 6$ we obtain a probability

of $\frac{1}{36}$. There are three cases in which the sum is 4, namely $(1,3)$, $(2,2)$, and $(3,1)$, thus

$$\mathbf{P}(X = 4) = \tfrac{3}{36} = \tfrac{1}{12}.$$

The probability mass function uniquely determines how the probability of a random variable is distributed. In particular, it defines its *distribution function*.

Definition 1.14 ((probability) distribution function)**.** The *(probability) distribution function*, or simply *distribution*, of a random variable X is the function $F : \mathbf{R} \to [0,1]$ given by

$$F(x) = \mathbf{P}(X \leq x).$$

The distribution function is monotonically increasing, $\lim_{x\to-\infty} F(x) = 0$ and $\lim_{x\to\infty} F(x) = 1$, and it is right-continuous, i.e., $F(x+h) \to F(x)$, as $h \searrow 0$.

An important distribution, which is often considered in this thesis, is the *binomial distribution* with parameters n and p. It is used to reflect the number of successes in n independent trials with a probability of success of p. A random variable X is binomially distributed with parameters n and p, abbreviated by $\mathrm{Bin}(n,p)$, if its probability mass function is given by

$$\mathbf{P}(X = k) = \tbinom{n}{k} p^k (1-p)^{n-k}.$$

Imagine we are throwing a fair coin n times, then the total number of heads is distributed as $\mathrm{Bin}(n,p)$ with $p = 1/2$.

Using the concept of probability mass functions, the definition of mutually independence can be adapted to random variables. The random variables $X_1, \ldots, X_n$ are called mutually independent if

$$\mathbf{P}\left(\textstyle\bigcap_{i\in L}(X_i = x_i)\right) = \prod_{i\in L} \mathbf{P}(X_i = x_i)$$

for each subset $L \subseteq \{1,\ldots,n\}$, and all values $x_1, \ldots, x_n \in \mathbf{R}$. If several mutually independent random variables X_i, $i = 1,\ldots,n$, have the same distribution function, one says that the X_i are *independent, identically distributed* (iid) random variables.

A basic characteristic of a random variable X is its *expectation*, also referred to as first moment of X. It reflects the variable's average value weighted by the probability. For a discrete random variable X its expectation is denoted by $\mathbf{E}\,X$ and defined as

$$\mathbf{E}\, X := \sum_i i\, \mathbf{P}(X = i).$$

The summation is over all values in the range of X and it is only finite if $\sum_i |i|\, \mathbf{P}(X = i)$ converges, otherwise it is unbounded. Note that the kth moment of a random variable X is given by $\mathbf{E}(X^k) = \sum_i i^k\, \mathbf{P}(X = i)$. In our example with the two dice we obtain the expected value of X as

$$\mathbf{E}\, X = 2 \cdot \tfrac{1}{36} + 3 \cdot \tfrac{2}{36} + 4 \cdot \tfrac{3}{36} + \ldots + 12 \cdot \tfrac{1}{36} = 7.$$

For a binomially distributed variable X with parameters n and p, $\mathbf{E}\, X = np$. Since the expectation is defined as a sum, it is obvious that it behaves linearly for discrete (not necessarily independent) random variables X_i, $i = 1, \ldots, n$, and a constant c:

$$\mathbf{E}\left(\sum_{i=1}^n X_i\right) = \sum_{i=1}^n \mathbf{E}\, X_i \qquad \text{and} \qquad \mathbf{E}(cX_i) = c\, \mathbf{E}\, X_i.$$

In the analysis of random variables one is often interested in bounding the so called tail probability of a random variable X, that is, the probability that X exceeds a real number $\alpha > 0$. The expectation is a useful measure for this and yields with $r \in \mathbf{Z}_+$ the following inequality, often referred to as *Markov inequality*

$$\mathbf{P}(|X| \geq \alpha) \leq \frac{\mathbf{E}\,(|X|^r)}{\alpha^r}. \tag{1.2}$$

Advanced concepts Just as the conditional probability, one can also define the *conditional expectation* of a random variable Y as

$$\mathbf{E}(Y \mid Z = z) := \sum_y y\, \mathbf{P}(Y = y \mid Z = z),$$

where the sum is over all y in the range of Y, and Z is a random variable. As for the conditional probability, $\mathbf{E}(Y \mid Z = z) = E(Y)$ if Y and Z are independent. If no value for Z is specified, $\mathbf{E}(Y \mid Z)$ is itself a random variable $f(Z)$ taking values $\mathbf{E}(Y \mid Z = z)$ for $Z = z$.

Proposition 1.15. *The conditional expectation has the following properties for random variables X, Y, and Z:*

(i) $\mathbf{E}\, Y = \sum_z \mathbf{E}(Y \mid Z = z)\, \mathbf{P}(Z = z)$;

(ii) $\mathbf{E}\, Y = \mathbf{E}(\mathbf{E}(Y \mid Z))$;

(iii) $\mathbf{E}(aX + bY \mid Z) = a\,\mathbf{E}(X \mid Z) + b\,\mathbf{E}(Y \mid Z)$ *for* $a, b \in \mathbf{R}$;

(iv) $\mathbf{E}\,Y = \mathbf{E}(Y \mid Z) \cdot \mathbf{P}(Z) + \mathbf{E}(Y \mid \overline{Z}) \cdot \mathbf{P}(\overline{Z})$ *with complement* $\overline{Z}$ *of* Z*; and*

(v) $\mathbf{E}(Y \mid Z) = \mathbf{E}\,(\mathbf{E}(Y \mid X, Z) \mid Z) = \mathbf{E}\,(\mathbf{E}(Y \mid Z) \mid X, Z)$.

When dealing with sums of independent random variables it is useful to exploit their representation via *probability generating functions*, for which we follow [GS92]. The probability generating function reflects a sequence of probabilities just as the general generating function reflects a sequence of real numbers. Let X be a discrete random variable taking values in the nonnegative integers $\{0, 1, 2, \ldots\}$. Its distribution is defined by the sequence of probabilities $f(i) = \mathbf{P}(X = i)$ for $i = 0, 1, \ldots$. Then the probability generating function G of the random variable X is defined as

$$G_X(s) = \mathbf{E}(s^X) = \textstyle\sum_{i=0}^{\infty} s^i\,\mathbf{P}(X = i).$$

G_X converges at least when $|s| \leq 1$, and in its radius of convergence it is unique with respect to $f(i)$. For a generating function $G(s) = \sum_{i=1}^{\infty} a_i s^i$, *Abel's Theorem* states that, if $a_i \geq 0$ for all i and $G(s)$ is finite for $|s| < 1$, then $\lim_{s\uparrow 1} G(s) = \sum_{i=0}^{\infty} a_i$, independent of whether the sum is finite or not. Hence, $\lim_{s\to 1} G_X(s) = \sum_{i\geq 0} f(i)$ since the conditions of Abel's Theorem are always fulfilled by a probability generating function. Further, $G_X(0) = \mathbf{P}(X = 0)$ and $G_X(1) = 1$. In general, probability generating functions have the following properties.

Theorem 1.16. *Let all random variables named herein be nonnegative and discrete, and let* G_* *be the probability generating function of the random variable* $*$*. Then*

(i) for a random variable X*, its expected value is*

$$\mathbf{E}(X) = \frac{d}{ds} G_X(1);$$

(ii) if the two random variables X *and* Y *are independent, then*

$$G_{X+Y}(s) = G_X(s) G_Y(s);$$

(iii) let $X_i, i = 1, 2, \ldots$*, be a sequence of iid random variables with common probability generating function* G_X*, and let* $N \geq 0$ *be a random variable*

that is independent of the X_i. Then the probability generating function G_Z of the random variable $Z = X_1 + \ldots + X_N$ is given by

$$G_Z(s) = G_N(G_X(s)).$$

The theorem shows the usefulness of generating functions when it comes to compositions of random variables. See [GS92] for more details.

Random processes We now turn to collections of random variables.

Definition 1.17 (random process)**.** A *random process* is a collection $\mathbf{X} = \{X(t) \mid t \in T\}$ of random variables $X(t)$ for all $t \in T$, where $T \subseteq \mathbf{R}$ is an arbitrary set.

A random process models the value of a random variable over time. The random variable $X(t)$ is the state of the process at time t. If $X(t)$, $t \in T$, assumes values from a countably finite set, $\mathbf{X}$ is a *discrete space* process. If T is countably finite, $\mathbf{X}$ is a *discrete time* process. Herein we focus on discrete time and space processes that are *Markovian*, i.e., which satisfy

$$\mathbf{P}(X(t) = a_t \mid X(t-1) = a_{t-1}, \ldots, X(0) = a_0) = \mathbf{P}(X(t) = a_t \mid X(t-1) = a_{t-1}).$$

The state at time t depends only on the previous state, but it is independent of the history how the $(t-1)$th state was reached. A special class of random processes are martingales.

Definition 1.18 (martingale, stopping time)**.** A sequence of random variables $Z_0, Z_1, \ldots$ is a *martingale* with respect to the sequence $X_0, X_1, \ldots$ if, for all $n \geq 0$, the following conditions hold:

(i) Z_n is a function of $X_0, X_1, \ldots, X_n$;

(ii) $\mathbf{E}(|Z_n|) < \infty$; and

(iii) $E(Z_{n+1} \mid X_0, \ldots, X_n) = Z_n$.

A nonnegative, integer valued random variable T is a *stopping time* for the sequence $\{Z_n, n \geq 0\}$ if the event '$T = t$' depends only on the random variables $Z_0, \ldots, Z_t$.

A martingale can have a finite or a countably infinite number of elements. A stopping time is a time at which the random process can be stopped but it must only depend on the previous time steps. Imagine

there is a gambler playing repeatedly one game. A stopping time for him is, for example, the *first* time that he wins five times in a row while the time when he wins five times in a row for the *last* time is no stopping time as it depends on the future. The main theorem for martingales is the following.

Theorem 1.19 (Martingale Stopping Theorem)**.** *If $Z_0, Z_1, \ldots$ is a martingale with respect to $X_0, X_1, \ldots$ and if T is a stopping time for $X_0, X_1, \ldots$, then*

$$\mathbf{E}(Z_T) = \mathbf{E}(Z_0),$$

whenever one of the subsequent conditions holds:

- *the Z_i are bounded, i.e., there exists a constant $c \geq 0$ with $|Z_i| < c$, for all i;*
- *T is bounded; or*
- *$\mathbf{E}(T) < \infty$, and there is a constant $c \geq 0$ such that $\mathbf{E}(Z_{i+1} - Z_i \mid X_0, \ldots, X_i) < c \; \forall i$.*

With this we close our recapitulation of probability theory and refer to the named introductory literature for details.