# Chapter 2

# Preliminaries

In this chapter we give the necessary notations and basic definitions that we use throughout the thesis.

## Complexity-theoretic Notations and Definitions

We fix the finite alphabet $\Sigma = \{0,1\}$. $\Sigma^*$ denotes the set of strings over $\Sigma$. A subset of $\Sigma^*$ is called a language. $\Sigma^n$ ($\Sigma^{\leq n}$) denotes the set of strings over $\Sigma$ of length $n$ (respectively, $\leq n$). $\lambda$ denotes the empty string. For an $x \in \Sigma^*$, $|x|$ denotes the length of $x$. For $y_1, y_2 \in \Sigma^*$ $y_1 \leq y_2$ denotes that $y_1$ is lexicographically smaller than or equal to $y_2$. For any finite set $X$, $|X|$ denotes the cardinality of $X$. Note that we are using the same notation for the cardinality of a set and length of a binary string. The meaning of the notation will be clear from the context. For $L \subseteq \Sigma^*$, $0L$ (1L) denotes the language obtained by prefixing 0 to all the strings in $L$ (respectively 1). For sets $A$ and $B$, symmetric difference of $A$ and $B$ is denoted by $A \triangle B$. The base-2 logarithm function is denoted as log. We use some standard pairing functions, denoted by $\langle \cdot, \cdot \rangle$ which can be computed and inverted in polynomial time. $\mathbf{Z}$ and $\mathbf{N}$ denote the set of integers and natural numbers respectively. Let $X = \{x_1, \ldots, x_n\}$ be an ordered set. Then $(l_{x_1}, \ldots, l_{x_n})$ denotes a formal tuple indexed by set $X$. Some times we denote such a tuple by $(l_x | x \in X)$.

Next we very briefly describe the framework of computation we will be working with. For detailed definitions, we refer the reader to standard text books like [BDG88, Pap94].

The standard Turing machine will be our basic model of computation. We will consider both deterministic and nondeterministic Turing machines. Sometimes we allow Turing machines to have access to an oracle. In this case Turing machines will have an additional write-only tape and a special state called QUERY-state, which are used in a standard way to access the oracle. We assume the familiarity with the standard complexity classes such as P, NP, PH, FP, #P. Refer [Pap94] for the definitions of these classes. For any complexity class $\mathcal{L}$, Co-$\mathcal{L}$ denotes the class of languages whose complement is in $\mathcal{L}$.

Fenner et. al., in [FFK94] defined the class of functions GapP as the closure of Valiant's class #P under subtraction. A function $f : \Sigma^* \to \mathbf{Z}$ is *gap-definable* if there is a polynomial-time nondeterministic (in short, NP) machine $M$ such that, for each $x \in \Sigma^*$, $f(x)$ is the difference between the number of accepting paths (denoted by $acc_M(x)$) and the number of rejecting paths (denoted by $rej_M(x)$) of $M$ on input $x$. For each NP machine $M$ let $gap_M$ denote the gap-definable function defined by it. Let GapP denote the class of gap-definable functions. It is easy to see that GapP is the closure of #P under subtraction. The class GapP enjoys nice closure properties. Most of these closure properties have been shown in [FFK94]. Many of the counting classes that have been studied in the literature have equivalent characterizations using gap-definable functions.

Now we give the definitions of three counting classes which are of main importance to us.

**Definition 2.0.1** A language $L$ is in

- UP if there is an NP machine $M$ such that: $x \in L$ implies that $acc_M(x) = 1$, and $x \notin L$ implies that $acc_M(x) = 0$.

- SPP if there is an $f \in$ GapP such that: $x \in L$ implies that $f(x) = 1$, and $x \notin L$ implies that $f(x) = 0$.
- LWPP if there are functions $f \in$ GapP and $h \in$ FP such that: $x \in L$ implies that $f(x) = h(0^{|x|})$, and $x \notin L$ implies that $f(x) = 0$.

It follows from the definitions that UP $\subseteq$ SPP $\subseteq$ LWPP.

For the sake of completeness, we give definitions of PP, $\mathrm{C_{=}P}$ and $\mathrm{Mod}_k\mathrm{P}$. A language $L$ is in PP if there is a GapP function $f$ such that: $x \in L$ iff $f(x) > 0$. A language $L$ is in $\mathrm{C_{=}P}$ if there are functions $f \in$ GapP and $h \in$ FP such that: $x \in L \Leftrightarrow f(x) = h(0^{|x|})$. A language $L$ is in $\mathrm{Mod}_k\mathrm{P}$ if there is function $f \in$ GapP such that $x \in L \Leftrightarrow f(x) = 0 \pmod{k}$.

For all the above-mentioned classes, their relativized versions can be defined by allowing the nondeterministic machines accepting the languages in the class to have access to an oracle. If $M$ is an NP machine which accesses an oracle $A$, then the corresponding *gap* function is denoted by $gap_{M^A}$. For any class $\mathcal{C}$ which is relativizable, $\mathcal{C}^A$ denotes the relativized version of $\mathcal{C}$ with respect to the set $A$. Then a language $L$ is said to be *low* for $\mathcal{C}$ if $\mathcal{C}^A = \mathcal{C}$.

## Group Theory: Notations, Definitions and Basic Results

Here we give some notations and basic definitions from group theory. We also describe some basic results. For further results and their proofs, please refer to standard textbooks [Bur55, Hal59].

A *group* is a tuple $\langle G, * \rangle$ where $G$ is a nonempty set and $*$ is a binary operation (we call this operation "product") on $G$ satisfying the following properties. $G$ is closed under $*$. The operation $*$ is associative. There is an element $e \in G$, called the *identity* of $G$, such that $x * e = e * x = x$, for all $x \in G$. For every $x \in G$ there exists a unique $x^{-1} \in G$, called the *inverse* of $x$, such that $x * x^{-1} = x^{-1} * x = e$.

When there is no ambiguity we do not explicitly specify the group operation and denote the product $x * y$ by $xy$. Also, by abusing notation, henceforth we denote the group $\langle G, * \rangle$ by just $G$.

Let $G$ be a group. A subset $H$ of $G$ is called a *subgroup* of $G$ (denoted $H < G$ or $G > H$) if $H$ is a group under the group operation of $G$. For a subset $S$ of $G$, the smallest subgroup of $G$ containing $S$ is called the group *generated by* $S$ and is denoted by $\langle S \rangle$ (note that we are using the same notation for the pairing function also. The exact meaning of the notation will be clear from the context). This group is the same as the set of all finite products of elements from $S$. A subset $S$ of $G$ is a generator set for $G$ if $G$ is identical to $\langle S \rangle$. A group $G$ is *finite* if the cardinality of the set $G$ is finite. In this thesis, we are only interested in finite groups. Henceforth by a group we mean a finite group. The *order* of $G$ is defined as the cardinality of the set $G$ and is denoted by $|G|$. A group is said to be cyclic if it is generated by a single element. For an element $g \in G$, the *order* of $g$ (denoted as $o(g)$) is the order of the cyclic subgroup generated by $g$. This is the same as the smallest positive integer $k$ such that $g^k = e$, ($g^k$ denotes the product of $g$, $k$ times) where $e$ is the identity of $G$.

A fundamental theorem in finite group theory, due to Lagrange, states that if $H < G$ then $|H|$ divides $|G|$. This theorem has a large number of algorithmic applications. For example, it follows from Lagrange's theorem that any finite group $G$ is generated by a set of group elements of cardinality bounded by $\log |G|$.

Let $H$ be a subgroup of a group $G$. For $g \in G$ the set $\{hg \mid h \in H\}$, denoted by $Hg$, is called a *right coset* of $H$ in $G$. Similarly, the set $gH = \{gh \mid h \in H\}$ is called a *left coset* of $H$ in $G$. $H$ is a *normal* subgroup of $G$ if for all $g \in G$ it holds that $Hg = gH$. A fundamental result in the theory of groups is that if $H$ is a normal subgroup of $G$, then the set of right cosets of $H$ in $G$ forms a group (called the *factor group* or a *quotient group* induced by $H$ and denoted by $G/H$) under the binary operation $\cdot$ defined as $Hx \cdot Hy = Hxy$. The identity element of this group is the coset $He = H$. For a set $X \subseteq G$, the *normal closure* of $X$ is the smallest normal subgroup containing $X$.

Two groups $G$ and $H$ are said to be isomorphic if there is a bijection (set-theoretic) $\phi$ from $G$ to $H$ such that for any $x, y \in G$, $\phi(xy) = \phi(x)\phi(y)$. Isomorphism preserves all structural properties of groups.

Let $p$ be a prime. A $p$-group is a finite group whose order is a power of $p$. Let $G$ be finite group such that $|G| = p_1^{e_1} p_2^{e_2} \dots p_r^{e_r}$. The existence of subgroups in $G$ which are $p$-groups, is given by Sylow's theorem. That is, for each $i$ there is a subgroup of $G$ of order $p_i^{e_i}$. A subgroup of $G$ of order $p_i^{e_i}$ is referred to as a *$p_i$-Sylow subgroup* of $G$.

Let $(X, *)$ and $(Y, .)$ be two groups. The *direct product* of the groups $X$ and $Y$ is defined as the group $(X \times Y, \circ)$, where $X \times Y$ is the cartesian product of sets $X$ and $Y$, and for $(x_1, y_1), (x_2, y_2) \in X \times Y$ their $\circ$ composition is defined as $(x_1, y_1) \circ (x_2, y_2) = (x_1 * x_2, y_1.y_2)$. Let $H, K$ be subgroups of a group $G$. Suppose that $H$ is normal in $G$ and the set $\{xy \mid x \in H, y \in K\} = G$. Then $G$ is isomorphic to the direct product $H \times K$.

## Solvable groups

Here we give the definition of a *solvable* group and state some properties of them. Intuitively solvable groups can be thought of as a generalization of *abelian groups.* A group $G$ is *abelian* if $\forall x, y \in G : xy = yx$; that is $xyx^{-1}y^{-1} = e$. In general, the element $xyx^{-1}y^{-1}$ is called the *commutator* of elements $x$ and $y$ in $G$. The subgroup of $G$ generated by the set $\{xyx^{-1}y^{-1} \mid x, y \in G\}$ is called the *commutator subgroup* of $G$. We denote this subgroup by $G'$. Observe that if $G$ is abelian, $G'$ is the trivial group containing only the identity element. The commutator subgroup $G'$ is actually a normal subgroup of $G$ and the factor group $G/G'$ is abelian. For a group $G$, the sequence $G = G_0 > G_1 > \dots$ is called the commutator sequence, where each group $G_i$ is the commutator subgroup of $G_{i-1}$. $G$ is solvable if the commutator sequence terminates in the trivial subgroup $\langle e \rangle$ in finitely many steps. This intuitively means that any solvable group can be decomposed into a series of abelian factor groups.

Solvable groups form a large subclass of all finite groups. In fact, a celebrated result due to Fiet and Thompson says that any finite group of odd order is solvable. Any subgroup of a solvable group is solvable. It follows from Lagrange's theorem that, if $G$ is solvable, then length of the commutator sequence is bounded by $\log |G|$. From a computational viewpoint this fact is very useful. It also holds that two solvable groups $G$ and $H$ are isomorphic if and only if the factor group $H_{i-1}/H_i$ is isomorphic to $G_{i-1}/G_i$ for all $i$. Here $H_i$ ($G_i$) is the $i^{\text{th}}$ element in the commutator series of $H$ (respectively, $G$).

## A Complexity-theoretic Technique

The main complexity-theoretic technique that we use for showing membership in SPP is the following lemma proved in [KST92]. In fact, in [KST92] a more general version of this lemma is proved. For our purposes the following is enough. For the statement of this lemma, we first give some definitions.

Let $M$ be an oracle NP machine and let $A \in$ NP be accepted by an NP machine $N$. We say that a query $y$ made by $M$ is *1-guarded* for $N$ if $N(y)$ has at most 1 accepting path. We say that $M$ makes 1-guarded queries to $A$, if there exists an NP machine $N$ accepting $A$ such that on all the inputs $x$ to $M$, the queries made by $M$ to $A$ are 1-guarded for $N$.

**Lemma 2.0.2** [KST92] *Let $M$ be a nondeterministic polynomial-time oracle machine that makes 1-guarded queries to $A \in$ NP. Then the function $gap_{M^A}(x)$ is in* GapP.

*Proof.* Let $x$ be an input to $M$ which makes 1-guarded queries to $A \in$ NP. Let $N$ be the corresponding NP machine accepting $A$. For a computation path $\rho$ of $M$, define $v(\rho) = 1$ if $\rho$ is accepting and $v(\rho) = -1$ if $\rho$ is rejecting. In this notation the gap function defined by $M$ is $gap_{M^A}(x) = \Sigma_\rho v(\rho)$. Let $q$ be the polynomial bounding the number of queries asked by $M$. Without loss of generality, we can assume that on any input $x$, $M$ makes exactly $q(|x|)$ queries. Design a machine $M'$ as follows. $M'$ on input $x$ guesses a computation path $\rho$ of $M$ and simulates $M$ on $\rho$. For the $i^{\text{th}}$ query $y_i$, made by $M$, $M'$ guesses $a_i \in \{0,1\}$ as an answer to the query (this way $M'$ avoids access to the oracle) and continues simulation of $M$ on $\rho$, treating $a_i$ as the answer to the query $y_i$ of $M$. At the end of the computation on $\rho$, $M'$ produces the following *gap*.

$$gap \;=\; v(\rho)\Pi_{i=1}^{q(|x|)}(a_i gap_N(y_i) + (1-a_i)(1-gap_N(y_i))) \qquad (2.1)$$

We argue that $gap_{M'}(x) = gap_{M^A}(x)$ as follows. Call those computation paths of $M'$, where all the guessed answers $q_i$ to the queries $y_i$ are correct ($a_i = 1$ if and only if $y_i \in A$), *good*. Observe that corresponding to any computation path $\rho$ of $M$, there is exactly one computation path $\rho'$ of $M'$ which is *good*. We show that, on the *good* paths, $M'$ produces a $gap = v(\rho)$ and on all the computation paths which are not *good*, $M'$ produces a $gap = 0$. From this and the observation that for any computation path $\rho$ of $M$, there is exactly one computation path $\rho'$ which is *good*, it will follow that $gap_{M'}(x) = gap_{M^A}(x)$.

Consider a path $\rho'$ of $M'$ which is *good*. Since $\rho'$ is *good*, all the queries constructed by $M'$ will be same as the queries made by $M$ on the path $\rho$ and hence will be 1-guarded for $M$. Consider the $i^{\text{th}}$ term in the product of RHS of (2.1). If $a_i = 1$, then $y_i \in A$ and $gap_N(y) = 1$ (since the query is 1-guarded for $N$). If $a_i = 0$ then $y_i \notin A$ and $gap_N(y) = 0$. Hence on the *good* paths, $M'$ produces a $gap = v(\rho)1^{q(|x|)} = v(\rho)$.

Now we shall consider the gap produced by $M'$ on paths which are not *good*. Let $i'$ be the first value of $i$ for which $M'$ guesses $a_i$ which is not correct. This means that $a_{i'} = 0$ if an only if $y_{i'} \in A$. Here we make a crucial observation that all the queries $y_j$ for $j \leq i'$, computed by $M'$, as queries made by $M$ to $A$ are 1-guarded for $N$ (the queries $y_k$ computed by $M'$ for $k > i'$ may not be 1-guarded for $N$ since the answer to query $y_{i'}$ is guessed wrong by $M'$). Now if $a_{i'} = 1$, $gap_N(y_{i'}) = 0$ and hence $a_i gap_N(y_i) + (1 - a_i)(1 - gap_N(y_i)) = 0$. Similarly if $a_{i'} = 0$, $gap_N(y_{i'}) = 1$ and $a_i gap_N(y_i) + (1 - a_i)(1 - gap_N(y_i)) = 0$. In any case, contribution of the $i'^{\text{th}}$ term in the product of RHS of 2.1 is 0 and hence the total gap produced by $M'$ on $\rho$ is 0. This concludes the proof of the lemma. ■

The above lemma is useful in showing certain problems to be in the class SPP. For example, to show that a language $B$ is in SPP, it is enough to show that there exists a polynomial-time deterministic machine $M$ and a language $A \in$ NP such that $M$ makes 1-guarded queries to $A$ and accepts $B$. To see this, observe that the machine $M$ can be very easily modified to get a machine $M'$ such that $M'$ will have a *gap*=1 if $x \in B$ and will have a *gap*=0 otherwise. Now we can apply the above lemma directly to get $B \in$ SPP. For all the upper bound proofs in Chapter 4, we will be explicitly using this method for proving membership in SPP. On the other hand, in Chapter 5, we will essentially follow the line of proof of the above lemma for showing upper bound on learning linear spaces using an SPP-assistant. The above lemma can be generalized for getting a sufficient condition for membership in the class LWPP. This is done essentially by generalizing the notion of 1-guarded queries to $f$-guarded queries, where $f$ is a function on natural numbers. In fact, a slightly more complicated adaptation of the above proof is used for proving upper bound on the learnability of permutation groups using an LWPP-assistant in Chapter 5. By slightly modifying the above lemma we can also show that SPP is low for the classes PP, $\mathrm{C_{=}P}$ and $\mathrm{Mod}_k\mathrm{P}$.