Chapter 4

Solvable Black-box Group Problems

4.1 Introduction

This chapter is devoted to showing the membership of some basic problems from computational group theory in the class SPP. For the sake of completeness, we once again give the exact definitions of problems of interest to us.

Let $\mathcal{B} = \{B_m\}_{m>0}$ be a group family. The problems of interest to us are the following:

Membership Testing $\triangleq \{\langle m, S, g\rangle \mid \langle S\rangle < B_m \text{ and } g \in \langle S\rangle\}$.
Order Verification $\triangleq \{\langle m, S, n\rangle \mid \langle S\rangle < B_m \text{ and } |\langle S\rangle| = n\}$.
Group Isomorphism $\triangleq \{\langle m, S_1, S_2\rangle \mid \langle S_1\rangle \text{ and } \langle S_2\rangle \text{ are isomorphic subgroups of } B_m\}$.

We show that these problems over any group family are in the class SPP when restricted to solvable groups. For showing this upper bound, we design appropriate deterministic oracle algorithms accepting the above languages which make only 1-guarded queries to a language in NP. Then by Lemma 2.0.2 proved in Chapter 2 we get the desired upper bound.

Remark. Note that we are considering these problems when the groups involved in the definitions of the problems are solvable. Although there is no deterministic polynomial time algorithm known for Solvability Testing over arbitrary black-box groups, later in this chapter we show that Solvability Testing is in SPP. Hence, without loss of generality we can assume that the groups encoded in the problem instances are solvable.

Let $\mathcal{B} = \{B_m\}_{m>0}$ be a group family. Recall the definition of a solvable group. A group G is said to be solvable if the commutator sequence $G = G_0 > G_1 > \ldots$ terminates in the trivial subgroup $\langle e \rangle$ in finitely many steps. Let k be the length of this series. From Lagrange's theorem, it follows that k is bounded by a polynomial in m. Now, since G_{i-1}/G_i is abelian for all $i \leq k$, we can informally say that any solvable subgroup of B_m can be viewed as a short series of abelian factor groups. So intuitively it is clear that abelian groups are the basic building blocks of solvable groups and it is natural that tight upper bounds on abelian group problems may lead to upper bounds for solvable groups. In view of this observation, in the next section we concentrate on oracle algorithms for abelian groups. As an illustration, at the end of the next section we give an upper bound of SPP for the problems over abelian groups. Then, we extend these upper bounds to more general class of solvable groups.

4.2 Managing Abelian Groups

For dealing with abelian groups, we use a constructive version of the fundamental theorem about the structure of finite abelian groups. This theorem completely characterizes any finite abelian group up to isomorphism. We next state this theorem. Proof of this theorem can be seen in any standard book in group theory (see for example [Hal59, Bur55]).

Theorem 4.2.1 [Bur55] *Let G be a finite abelian group such that $|G| = p_1^{e_1} p_2^{e_2} \dots p_r^{e_r}$, where the p_i's are distinct primes.*

1. *The group G can be expressed as the direct product of its Sylow subgroups $S(p_1), S(p_2), \dots, S(p_r)$ where $|S(p_i)| = p_i^{e_i}$ for $1 \leq i \leq r$.*
2. *For $1 \leq i \leq r$, each Sylow subgroup $S(p_i)$ can be uniquely expressed as the direct product of cyclic groups of orders $p_i^{e_{i1}}, p_i^{e_{i2}}, \dots, p_i^{e_{is_i}}$ such that $e_{i1} \geq e_{i2} \geq \dots \geq e_{is_i}$ and $\Sigma_{j=1}^{s_i} e_{ij} = e_i$.*
3. *This decomposition of G is unique up to isomorphism.*

The above theorem implies that, for any abelian p-group G of order p^e, there is a *unique* sequence of natural numbers $e_1 \geq e_2 \geq \dots \geq e_m$ such that $\Sigma_{1 \leq j \leq m} e_j = e$ and G can be expressed as a direct product of m cyclic groups of respective orders p^{e_j}, $1 \leq j \leq m$. The sequence $(e_1, e_2, \dots, e_m)$ is called the *type* of the p-group G. Since any abelian group can be decomposed into its Sylow subgroups (Theorem 4.2.1, we can extend the definition of the type to an arbitrary abelian group.

A consequence of Theorem 4.2.1 is the existence of special kind of generator sets, called *independent* generator sets, for any abelian group. Let G be an abelian group which is a direct product of n cyclic groups $C_1, C_2, \dots, C_n$ generated by elements $g_1, g_2, \dots, g_n$ respectively. Then the set $S = \{g_1, g_2, \dots, g_n\}$ is a generator set for G. Since G is the direct product of $C_1, C_2, \dots, C_n$, it follows that for any $g \in S$, $\langle g \rangle \cap \langle S - \{g\} \rangle = \{e\}$. We call a generator set with this property an independent generator set. Formally, we have the following definition.

Definition 4.2.2 Let G be a finite abelian group. An element $g \in G$, $g \neq e$, is said to be *independent* of a set X of elements of G if $\langle g \rangle \cap \langle X \rangle = \{e\}$. A generator set S of G is an *independent generator set* for G, if every $g \in S$ is independent of $S - \{g\}$.

As a consequence of Theorem 4.2.1, all finite abelian groups have independent generator sets. Also, the size of the generator set is logarithmic in the size of the group. Now we prove a key group-theoretic proposition about independent generator sets. For stating this, first we give some definitions.

Let G be an abelian group generated by a set $S = \{g_1, \ldots, g_k\} \subseteq G$ (in the sequel, we shall assume that the elements of a generator set are ordered according to some fixed order. Wherever necessary, we will prescribe the order). Since G is abelian, for any element $g \in G$, there is an ordered tuple of $(l_1, \ldots, l_k)$ (we call this tuple exponents), $1 \leq l_i \leq o(g_i)$ for all $1 \leq i \leq k$ such that, $h = \prod_{i=1}^{k} g_i^{l_i}$. In most of our algorithms in this chapter, a major computation involved is that of exponents. In some places such computations have to be carried out for abelian factor groups also. Motivated by this, we give the following definition.

Definition 4.2.3 Let G be an abelian group generated by a set $S \subseteq G$. Let $X \subseteq S$. Then for $g \in G$, a tuple $(l_x | x \in X)$ is called an X-*exponent* of g with respect to S, if for all $x \in X$, $1 \leq l_x \leq o(x)$ and there exists indices $\{l_y \mid y \in S - X\}$ such that $g = \prod_{s \in S} s^{l_s}$.

In most places the set S of the above definition will be clear from the context. In this case we omit the reference to S and simply say X-*exponent*.

Now we prove a proposition which guarantees the uniqueness of exponents for independent generator sets. This property of independent generator sets is crucial for us.

Proposition 4.2.4 *Let G be an abelian group and S be an independent generator set for G. Then for any $g \in G$ and $X \subseteq S$, the X-exponent of g is unique.*

Proof. Notice that it is enough to show that for any $g \in G$, the S-exponent is unique. Then it will follow from the definition that, for any $X \subseteq S$, the X-exponent of g is also be unique. Let $S = \{g_1, \ldots, g_n\}$. Suppose for some $g \in G$ the S-exponent of g is not unique. Let $(l_1, \ldots, l_n)$ and $(l'_1, \ldots, l'_n)$ be two S-exponent of g such that

$l_i \neq l'_i$. Then $g = g_1^{l_1} \dots g_n^{l_n} = g_1^{l'_1} \dots g_n^{l'_n}$. Since G is abelian, this implies that $g_i^{l_i - l'_i} \in \langle S - \{g_i\} \rangle$ which is a contradiction to the fact that S is independent since $g_i^{l_i - l'_i} \neq e$. ∎

The usefulness of independent generator sets comes from the above proposition. To make this clear, suppose $S \subseteq B_m$ be a generating set for an abelian group. Also, assume that the orders of all the elements in the set S are given. Then, it is easy to design an NP machine which on input $\langle m, S, g \rangle$ tests whether $g \in \langle S \rangle$. The machine basically guesses an S-exponent of g and verifies in polynomial time that the guess is correct. Now if S is an independent generator set then the NP machine will have a *unique* accepting path if and only if $g \in \langle S \rangle$. This observation along with the fundamental theorem (Theorem 4.2.1) indicates that computing an independent generator set and the type of the abelian group may be sufficient for all the three problems of our interest. Hence our focus is to design a deterministic oracle algorithm for this computation. This algorithm is allowed to use an NP language as oracle with the promise that the queries made to this oracle are 1-guarded. The main functions of this NP language will be to aid the algorithm for exponent and order computations. Notice that the exponent computation is at least as hard as computing the discrete logarithm. The key factor that helps to make 1-guarded queries to these NP languages, is the above proposition.

In the next subsection, we concentrate on the construction of an independent generator set for an abelian group from an arbitrary generator set. For application to solvable groups, we will actually be interested in the construction of an independent generator set for abelian *factor* groups. But, for clarity of presenting the methods involved, we first give a detailed design of a deterministic oracle algorithm (we shall call it INDEPENDENTGENERATOR) which takes an arbitrary generator set for an abelian group and converts it into an independent generator set by making 1-guarded queries to an NP language. In next section, we explain how we can modify INDEPENDENTGENERATOR to incorporate the construction of an independent generator set for abelian factor groups.

4.2.1 Constructing an Independent Generator Set

This subsection is devoted to the proof of the following theorem.

Theorem 4.2.5 *Let $\mathcal{B} = \{B_m\}_{m>0}$ be a group family. Then there exists a deterministic oracle algorithm* INDEPENDENTGENERATOR *and a language $L_{ig} \in$ NP such that* INDEPENDENTGENERATOR *takes $\langle m, S\rangle$ as input, where $S \subseteq B_m$, and L_{ig} as oracle. It outputs $|\langle S\rangle|$, types of each Sylow subgroup of $\langle S\rangle$ and a set of independent generators for $\langle S\rangle$ if $\langle S\rangle$ is abelian and outputs* NOT ABELIAN *if $\langle S\rangle$ is not abelian. Furthermore,* INDEPENDENTGENERATOR *runs in time polynomial in $|\langle m, S\rangle|$, and makes only 1-guarded queries to L_{ig}.*

As mentioned before, for the construction of an independent generator set, we use a constructive version of the Theorem 4.2.1. So, before we go into the details of the design of INDEPENDENTGENERATOR, we briefly explain the steps involved in the proof of the fundamental theorem. The first part of the theorem follows from the Sylow theorems. The proof of the second part, is essentially the proof of the following technical lemma. This lemma gives a greedy strategy to compute an independent generator set from an arbitrary set of generators for any abelian p-group. A proof of this lemma is given in the appendix at the end of the thesis.

Lemma 4.2.6 ([Bur55]) *Let G be a finite abelian p-group. Let $g_1, g_2 \ldots, g_i$ be i independent elements of G of orders $p^{m_1}, p^{m_2}, \ldots, p^{m_i}$ respectively such that for all j, $1 \leq j \leq i$, the maximum order of any element in the factor group $G/\langle\{g_1, g_2 \ldots, g_j\}\rangle$ is $p^{m_{j+1}}$. Let $g'_{i+1}\langle\{g_1, g_2 \ldots, g_i\}\rangle$ be an element in the factor group $G/\langle\{g_1, g_2 \ldots, g_i\}\rangle$ of order $p^{m_{i+1}}$. Further, let $(x_1, x_2, \ldots, x_i)$ be the unique $\{g_1, g_2 \ldots, g_i\}$-exponent of $(g'_{i+1})^{p^{m_{i+1}}}$. Then $p^{m_{i+1}}$ divides x_j for $1 \leq j \leq i$. Let $y_j = x_j/p^{m_{i+1}}$ for $1 \leq j \leq i$. Then $g_{i+1} = g'_{i+1} g_1^{-y_1} g_2^{-y_2} \ldots g_i^{-y_i}$ is an element of G of order $p^{m_{i+1}}$ which is independent of $\{g_1, g_2 \ldots, g_i\}$.*

The first step for computing an independent generator set in INDEPENDENT-GENERATOR, is the construction of generator sets for all the Sylow subgroups of $\langle S\rangle$. Once we have the generator sets for each of the Sylow subgroups of $\langle S\rangle$, then we can use the method given by Lemma 4.2.6 to convert each of them into independent generator sets. The union of the independent generator sets of its Sylow subgroups forms an independent generator set for the group $\langle S\rangle$.

The next group-theoretic lemma shows how to construct generator sets for the Sylow subgroups from a generator set of an abelian group. We give a proof of this in the appendix.

Lemma 4.2.7 *Let G be a finite group of order n and let $p_1^{e_1}p_2^{e_2}\dots p_r^{e_r}$ be the complete prime factorization of n. Let H be an abelian subgroup of G generated by the set $\{g_1, g_2 \dots, g_s\} \subseteq H$. Then for each j, the p_j-Sylow subgroup of H is generated by $X_j = \{g_i^{(n/p_j^{e_j})} \mid 1 \le i \le s\}$.*

From the statement of Lemma 4.2.6, it is clear that the two major computations involved in the construction of an independent generator set are the exponent and the order computations. We use a language $L_{ig} \in$ NP precisely to aid INDEPENDENTGENERATOR for these computations. Before we give the exact definition of the language L_{ig}, we state a group-theoretic proposition which will give us a method for computing the order. Since we will be interested in computing the order in factor groups, we state the proposition for factor groups. See the appendix for a proof.

Proposition 4.2.8 *Let G, H and K be finite groups such that $H, K < G$ and K is a normal subgroup of H. Let $|G| = n = p_1^{e_1}\dots p_r^{e_r}$ be the unique prime factorization of the order of G. Then for any element hK in H/K, the order $o(hK)$ is of the form $p_1^{d_1}\dots p_r^{d_r}$, where for all i; $1 \le i \le r$, d_i is the smallest integer j such that $(h^{n/p_i^{e_i}})^{p_i^j} \in K$.*

This proposition essentially says that the computation of order can be done in polynomial time, provided we have oracles for membership testing and factorization. Notice that computing large powers can be done in polynomial-time using the method of "doubling"; by computing the squares successively. Next we define the language L_{ig} which INDEPENDENTGENERATOR will be using as oracle for order and exponent computations.

Oracles for INDEPENDENTGENERATOR

Here we give the precise definition of the NP language L_{ig}. We define L_{ig} as a disjoint union of two prefix languages L_1 and L_2 in NP.

Let L be a language in NP accepted by an NP machine M. The prefix language of L with respect to M is defined as follows: $\mathit{Prefix}_M(L) = \{\langle x, y\rangle \mid \exists z \in \Sigma^*$ such that yz is an accepting path of M on input $x\}$. It is easy to see that $\mathit{Prefix}_M(L) \in \mathrm{NP}$. Moreover, if $L \in \mathrm{UP}$, then $\mathit{Prefix}_M(L) \in \mathrm{UP}$.

We use the following theorem from [FK92] for defining L_1. Let PRIMES denote the language consisting of prime numbers encoded in binary. We use the following theorem from [FK92].

Theorem 4.2.9 ([FK92]) PRIMES $\in$ UP.

Now, consider the following language L_1'.

$$L_1' \triangleq \{\langle n, p_1, \ldots, p_k, e_1, \ldots, e_k\rangle \mid \forall i \leq k,\ p_i \in \mathrm{PRIMES};\ p_1 < \ldots < p_k;\ n = \prod_{i=1}^{k} p_i^{e_i}\}$$

From the facts that PRIMES $\in$ UP and for any integer there exists a unique prime factorization, it follows that $L_1' \in \mathrm{UP}$. Let M_1' be a UP machine accepting L_1'. Define $L_1 = \mathit{Prefix}_{M_1'}(L_1')$. We have $L_1 \in \mathrm{UP}$. Notice that it is easy to design a deterministic oracle polynomial-time machine which can compute the complete factorization of any integer by a prefix search using L_1 as oracle. We formally write these observations as a proposition.

Proposition 4.2.10 *The language* $L'_1 \in \mathrm{UP}$. *Let* M'_1 *be a* UP *machine accepting* L'_1. *Let* $L_1 = Prefix_{M'_1}(L'_1)$. *Then* $L_1 \in \mathrm{UP}$. *Moreover, there exists a polynomial-time deterministic oracle machine* FACTOR *which takes an integer as input and computes its complete prime factorization using* L_1 *as oracle.*

Now we will define the language L_2. L_2 is defined as a prefix language of another language L'_2 defined below. Let $\mathcal{B} = \{B_m\}_{m>0}$ be a group family.

$$L'_2 \triangleq \{\langle m, S, g\rangle \mid g \in B_m, S \subseteq B_m, \text{ and } \forall h \in S; \exists l_h \leq o(h) \text{ such that } g = \Pi_{h \in S} h^{l_h}\}$$

The following proposition essentially shows how to make 1-guarded queries to L'_2.

Proposition 4.2.11 *There exists an NP machine* M'_2 *witnessing* L'_2 *in* NP. *Moreover, for those inputs* $\langle m, S, g\rangle \in L'_2$ *such that* S *is an independent generator set for the abelian group* $\langle S\rangle$, M'_2 *will have a unique accepting path.*

Proof Sketch. Consider an NP machine M'_2 which on input $\langle m, S, g\rangle$, first computes $|B_m|$, then guesses a string $\langle |B_m|, p_1, \ldots, p_k, e_1, \ldots, e_k\rangle$ and verifies that $B_m = \prod_{i=1}^{k} p_i^{e_i}$ by simulating the UP machine M'_1 given by the previous proposition. Notice that, at the end of this computation, M'_2 will have the complete prime factorization of $|B_m|$ on a unique path. Next, using the factorization of $|B_m|$, M'_2 computes $o(h)$ for all $h \in S$ in polynomial-time. For this computation it uses the method given in Proposition 4.2.8. That is, it computes d_i for all $i; 1 \leq i \leq r$ where d_i is the smallest j such that $(g^{n/p_i^{e_i}})^{p_i^j} = e$. Since $1 \leq d_i \leq e_i$ and e_i is bounded by a polynomial in m, for every i, each d_i can be computed in time bounded by a polynomial in the length of the input. After this computation, M'_2 guesses indices l_h for each $h \in S$ such that $1 \leq l_h \leq o(h)$ and accepts if $g = \Pi_{h \in S} h^{l_h}$. Note that $(l_h | h \in S)$ is an S-exponent of g. Now, if S is an independent generator set for the abelian group $\langle S\rangle$, by Proposition 4.2.4, the S-exponent of g will be unique and hence only one of the guesses of M'_2 leads to acceptance. ∎

Before we proceed further, we make a few remarks about the language L_2'. Firstly, notice that if $\langle S\rangle$ is abelian then L_2' is same as Membership Testing (for general groups this is a "weak Membership Testing" oracle since $L_2' \subset$ Membership Testing). This can be a little confusing since one of our goals is testing membership. But, though we will be using this language as an oracle, we will be very careful about accessing this oracle. We will always make sure that whenever a query $\langle m, S, g\rangle$ is made to this oracle, S is an independent generator set for $\langle S\rangle$. This is acceptable for us because, from the above proposition, such queries will be 1-guarded.

Now define L_2 as $\mathit{Prefix}_{M_2'}(L_2')$. Then $L_2 \in$ NP. Let M_2 be the corresponding NP machine accepting L_2. Given g and an independent generator set S for the abelian group $\langle S\rangle$, using L_2 as oracle we can prefix search for the X-exponent of g, where $X \subseteq S$. For each such query made to L_2, M_2 will have at most one accepting path. Thus, if S is an independent generator set for an abelian group G, then for any $X \subseteq S$ and $g \in G$, the unique X-exponent of g can be computed in deterministic polynomial-time by making only 1-guarded queries to L_2. These observations we state as a lemma.

Lemma 4.2.12 *Let $L_2 = \mathit{Prefix}_{M_2'}(L_2')$. Then $L_2 \in$ NP. Moreover there exists a deterministic polynomial-time oracle algorithm* EXPONENT *which takes $\langle m, S, X, g\rangle$ as input and L_2 as oracle such that, if the input satisfies the promise that S is an independent generator set for the abelian group $\langle S\rangle$, $X \subseteq S$ and $g \in \langle S\rangle$, then* EXPONENT *outputs the unique X-exponent of g. Furthermore, for such inputs* EXPONENT *makes only 1-guarded queries to the oracle. The behavior of* EXPONENT *is unspecified if the input does not satisfy the promise.*

In the next lemma, we give a deterministic oracle algorithm ORDER for computing the order of any element in a factor group. ORDER uses the method described in the Proposition 4.2.8 for this computation. It uses the UP language L_1 for factorization. Again, Membership Testing is needed for computing the order in a factor group. But we will be computing the order in a factor group H/K only when an independent generator set for K is already computed. So once again we can use the language L_2' in a 1-guarded manner for testing membership.

Lemma 4.2.13 *Let $\mathcal{B} = \{B_m\}_{m>0}$ be a group family. Then there exists a deterministic polynomial-time oracle algorithm* ORDER *which takes as input $\langle m, g, X, Y\rangle$, and $0L_1 \cup 1L_2'$ as oracle such that, if the input satisfies the promise that $X, Y \subseteq B_m$, Y is a normal subgroup of X, $g \in \langle X\rangle$ and Y is an independent generator set for $\langle Y\rangle$, then* ORDER *outputs $o(g\langle Y\rangle)$. Furthermore, for such inputs* ORDER *makes only 1-guarded queries to the oracle. The behavior of* ORDER *is unspecified if the input does not satisfy the promise.*

We have defined the languages that we will be using as oracles for *all* the algorithms in this chapter. We can unify these languages into a single language L_{ig} by defining $L_{ig} = 0L_1 \cup 1L_2$ (disjoint union of L_1 and L_2). Notice that all the languages L_1, L_2' and L_2 are subsumed by L_{ig}. Sometimes, for clarity of presentation, we may split L_{ig} into its component languages.

The Algorithm INDEPENDENTGENERATOR

Now we are ready to give a formal description of INDEPENDENTGENERATOR and prove that it has the behavior as given in Theorem 4.2.5.

We first explain how the algorithm works and then give a formal description of it. Algorithm INDEPENDENTGENERATOR on input $\langle m, S\rangle$, first checks whether $\langle S\rangle$ is abelian. This can be done easily by checking whether $g_i g_j = g_j g_i$ for all $g_i, g_j \in S$. Then it computes $|B_m|$ and using the algorithm FACTOR as subroutine computes the complete factorization of $|B_m|$. Knowing the factorization $p_1^{e_1} p_2^{e_2} \ldots p_r^{e_r}$ of $|B_m|$, it can compute a generator sets for all the Sylow subgroups of $\langle S\rangle$ using the method given in Lemma 4.2.7. Let X_j denote the generator set thus constructed for the p_j-Sylow subgroup.

The next step of INDEPENDENTGENERATOR is to convert the generator sets of the Sylow subgroups into independent generator sets. Lemma 4.2.6 provides a greedy strategy for this. Consider the p_j-group $\langle X_j \rangle$. Let it be of type $(m_1, m_2, \ldots, m_s)$. Let $g_1, g_2 \ldots, g_i$ be i independent elements of $\langle X_j \rangle$ so far constructed such that g_k is of order $p_j^{m_k}$ for $1 \leq k \leq i$. Suppose we could get an element $g'_{i+1} \in \langle X_j \rangle$ such that $g'_{i+1}\langle \{g_1, g_2 \ldots, g_i\} \rangle$ is of maximum order $p^{m_{i+1}}$ in the factor group $\langle X_j \rangle / \langle \{g_1, g_2 \ldots, g_i\} \rangle$. Then by the method of Lemma 4.2.6, we can convert g'_{i+1} to $g_{i+1} \in \langle X_j \rangle$ of order $p^{m_{i+1}}$ which is independent of $\{g_1, g_2 \ldots, g_i\}$ as follows. First compute the $\{g_1, g_2 \ldots, g_i\}$-exponent $(x_1, x_2, \ldots, x_i)$ of $(g'_{i+1})^{p_j^{m_{i+1}}}$ using the algorithm EXPONENT as subroutine. Then compute $g_{i+1} = g'_{i+1} g_1^{-y_1} g_2^{-y_2} \ldots g_i^{-y_i}$ where $y_k = x_k / p^{m_{i+1}}$, for $1 \leq k \leq i$. Then by Lemma 4.2.6, g_{i+1} is an element of $\langle X_j \rangle$ of order $p^{m_{i+1}}$ which is independent of $\{g_1, g_2 \ldots, g_i\}$, there by incrementing the partial independent set by one.

For implementing the above method, we need to compute an element $g'_{i+1} \in \langle X_j \rangle$ and $p^{m_{i+1}}$ such that $g'_{i+1}\langle \{g_1, g_2 \ldots, g_i\} \rangle$ is of maximum order $p^{m_{i+1}}$ in the factor group $\langle X_j \rangle / \langle \{g_1, g_2 \ldots, g_i\} \rangle$. For this the following two propositions can be employed.

The first proposition is an observation which is crucial for our algorithm. It says that, for an abelian p-group, any generator set contains an element of maximum order among all elements of the group. So for getting an element of maximum order, we need only to search in the generator set.

Proposition 4.2.14 *Let G be an abelian p-group generated by a set S. Let $(m_1, m_2, \ldots, m_s)$ be the type of G. If $g \in S$ is an element of maximum order among the elements of S, then g is of order p^{m_1}.*

Proof. G is a group of prime power order. Since $(m_1, m_2, \ldots, m_s)$ is the type of G, the maximum possible order of an element in G is p^{m_1}. If all elements of S are of order less than p^{m_1} then every element of G will be of order less than p^{m_1} since S

generates G. Thus g, which is of maximum order in S, must be of order p^{m_1}. ■

The next proposition allows us to apply the above observation for abelian factor groups. It follows from the group-theoretic fact that for any group G, G is isomorphic to $G/H \times H$ where H is a normal subgroup of G.

Proposition 4.2.15 *Let G be an abelian p-group of type $(m_1, m_2, \ldots, m_s)$. Let $H < G$ such that H is of type $(m_1, m_2, \ldots, m_i)$. Then G/H is of type $(m_{i+1}, m_{i+2}, \ldots, m_s)$.*

Finally, we give a formal description of the algorithm INDEPENDENTGENERATOR. We then argue that the behavior of the algorithm is as given in the Theorem. This will complete the proof of Theorem 4.2.5.

It is clear from the above discussions and the description of the algorithm that INDEPENDENTGENERATOR on input $\langle m, S\rangle$ outputs NOT ABELIAN if $\langle S\rangle$ is not abelian and outputs an independent generator set, order and type of $\langle S\rangle$ otherwise, in time polynomial in $|\langle m, S\rangle|$.

Now, we will see that the algorithm makes only 1-*guarded* queries to L_{ig}. The algorithm makes oracle calls in lines 4,13,15 and 16. In *line*-4, it makes a call to the language L_1 through the subroutine FACTOR which is 1-guarded since $L_1 \in$ UP. Now, it is clear from the above discussion, that whenever the algorithm enters the **while**-loop of *line*-13,the set S_i will be a set of independent elements. Hence, at *line*-13, the query $\langle m, S_i, h\rangle$ to L_2' will be 1-*guarded* from Proposition 4.2.11. The calls to L_{ig} at *lines* 15 and 16 are through the subroutines ORDER and EXPONENT. So these calls also will be 1-guarded since the generator set S_i involved in these subroutines calls are independent (see Lemma 4.2.12 and 4.2.13).

IndependentGenerator(m, S)
if $\exists g_i, g_j \in S$ such that $g_i g_j {g_i}^{-1} {g_j}^{-1} \neq e$
  then output NOT ABELIAN;
$n \leftarrow |B_m|$;
$p_1^{e_1} p_2^{e_2} \dots p_r^{e_r} \leftarrow$ Factor(n);
for $i : 1 \leq i \leq r$
do Compute the generators set X_i for the p_i-Sylow subgroup;
  /* `Using Lemma 4.2.7` */
end-for
for $i : 1 \leq i \leq r$
do $S_i \leftarrow \phi; N_i \leftarrow 1; T_i \leftarrow ()$
  /* S_i `stands for the set of independent generators`
  `of` p_i`-Sylow subgroup so far constructed` */
  while $\exists\ h \in X_i$ such that $\langle m, S_i, h\rangle \notin L_2'$
  do Find an element $h_j \in X_i$ such that
    $p_i^{l_j} \leftarrow$Order(m, h_j, X_i, S_i) is maximum;
    $(x_g \mid g \in S_i) \leftarrow$ Exponent(m, S_i, S_i, h_j);
    $S_i \leftarrow S_i \cup \{h_j \Pi_{g \in S_i} g^{-x_g / p_i^{l_j}}\}$;
    $N_i \leftarrow p_i^{l_j} . N_i$;
    $T_i \leftarrow T_i \cup (l_j)$;
  end-while
end-for
Output $\cup_{i=1}^{r} S_i$ as the independent generator set;
Output $\Pi_{1 \leq i \leq r} N_i$ as the order;
Output T_i as type of the p_i-Sylow subgroup of $\langle S\rangle$ for $1 \leq i \leq r$.

Upper Bounds for Abelian Group Problems

Here we shall illustrate how the algorithm IndependentGenerator can be modified to get upper bounds on the counting complexity of some of the problems over abelian black-box groups. Later in the chapter, we will extend these upper bounds for solvable groups.

Theorem 4.2.16 *Over any group family,* Membership Testing, Order Verification, *and* Group Isomorphism *for abelian black-box groups are in* SPP. *Hence these problems are low for* PP, $\mathrm{C_=P}$ *and the* $\mathrm{Mod}_k\mathrm{P}$ *for* $k \geq 2$.

Proof. Firstly, we can assume that the groups encoded in the problem instance are abelian for all the three problems. Now consider Membership Testing. Let $\mathcal{B}$ be a group family. Consider an oracle NP machine M_{mb} which on input $\langle m, S, g\rangle$ converts the set S to an independent generator set if S generates an abelian group, by simulating the algorithm INDEPENDENTGENERATOR on input $\langle m, S\rangle$. Let S' be the independent generator set thus constructed. Then M_{mb} makes one query $\langle m, S', g\rangle$ to the language L'_2 and it accepts, producing a *gap*=1, if the answer is 'YES'. If the answer to the query is 'NO' then M branches into an accepting and a rejecting path, thus producing a zero *gap*. So, from Theorem 4.2.5 and Proposition 4.2.11, it follows that M_{mb} runs in time polynomial in the length of the input and makes only 1-guarded queries to L_{ig}. Since $\langle m, S', g\rangle \in L'_2$ if and only if $g \in \langle S'\rangle = \langle S\rangle$, *gap* produced by M_{mb} is 0 if $g \notin \langle S\rangle$ and *gap* is 1 if $g \in \langle S\rangle$. Now from Lemma 2.0.2, it follows that Membership Testing is in SPP.

To show that Order Verification is in SPP, notice that one of the outputs of INDEPENDENTGENERATOR on input $\langle m, S\rangle$ is $|\langle S\rangle|$. Hence, it is easy to modify INDEPENDENTGENERATOR to an oracle algorithm M_o, which on input $\langle m, S, n\rangle$ produces zero *gap* if $n \neq |\langle S\rangle|$ and *gap* 1 if $n = |\langle S\rangle|$. Hence, again from Theorem 4.2.5 and Lemma 2.0.2 it follows that Order Verification is in SPP.

Finally we show that Group Isomorphism for abelian groups is in SPP. Let $\langle m, S_1, S_2\rangle$ be an instance of Group Isomorphism. S_1 and S_2 generate abelian subgroups of $B_m \in \mathcal{B}$. Firstly, observe that any two abelian p-groups are isomorphic if and only if their types are identical. This follows from the fact that the type uniquely determines an abelian p-group up to isomorphism [Bur55]. Furthermore, two abelian groups $\langle S_1\rangle$ and $\langle S_2\rangle$ are isomorphic iff $|\langle S_1\rangle| = |\langle S_2\rangle|$, and for every prime factor p of $|\langle S_1\rangle|$, the p-Sylow subgroup of $\langle S_1\rangle$ is isomorphic to the p-Sylow subgroup of $\langle S_2\rangle$. Thus, in order to check that $\langle S_1\rangle$ and $\langle S_2\rangle$ are isomorphic, it is only required to verify that the p-Sylow subgroups of $\langle S_1\rangle$ and $\langle S_2\rangle$ respectively have identical types, for each prime factor p of $|\langle S_1\rangle|$. Hence, from Theorem 4.2.5

and Lemma 2.0.2 it follows that Group Isomorphism is in SPP. ∎

4.3 Solvable Groups and Canonical Generator Sets

In the case of abelian groups, we have seen that the existence of an independent generator set played a crucial role in proving upper bounds for various computational problems. Motivated by this, we abstract out the essential structure that we need in a generator set which will help us in proving similar upper bounds for other classes of groups. More precisely we define the notion of *canonical generator set* for arbitrary classes of finite groups as a generalization to the notion of independent generator sets. Then we show the existence of canonical generator sets for the class of solvable groups.

Definition 4.3.1 Let $\mathcal{B} = \{B_m\}_{m>0}$ be any group family. Let $\mathcal{C}$ be a subclass of $\mathcal{B}$. The class of groups $\mathcal{C}$ has *canonical generator sets* if for every $G \in \mathcal{C}$, if $G < B_m$ then there is an *ordered* set $S = \{g_1, g_2, \ldots, g_s\} \subseteq G$ such that S generates G and each $g \in G$ can be uniquely expressed as $g = g_1^{l_1} g_2^{l_2} \ldots g_s^{l_s}$, where $1 \leq l_i \leq o(g_i), 1 \leq i \leq s$. Furthermore, $s \leq q(m)$ for a polynomial q. S is called a *canonical generator set* for G.

From the above definitions, it is clear that if S is an independent generator set for an abelian group G then each $x \in G$ can be uniquely expressed as a product $\Pi_{g \in S} g^{i_g}$ for $1 \leq i_g \leq o(g)$, and it holds that $|S| \leq \log(|G|)$. Thus for an abelian group, an independent generator set is a canonical generator set.

Note that the ordering of the elements of the canonical generator set is irrelevant for abelian groups since products commute. However, for classes of groups which are not abelian the ordering of elements in a canonical generator set has to be prescribed.

Now we shall extend the notion of exponents to any class of groups with canonical generator sets.

Definition 4.3.2 Let $\mathcal{B} = \{B_m\}_{m>0}$ be any group family. Let $\mathcal{C}$ be a subclass of $\mathcal{B}$ which has canonical generator sets. Let G be an element of $\mathcal{C}$. Let S be a canonical generator set for G and let $X \subseteq S$. Then for $g \in G$, the tuple $(l_x | x \in X)$ is called the X-*exponent* of g with respect to S if for all $x \in X$, $1 \leq l_x \leq o(x)$ and there exists indices $\{l_y \mid y \in S - X\}$ such that $g = \prod_{s \in S} s^{l_s}$.

From the definition it follows that for $g \in G$ the X-exponent of g is unique. We now show that the class of solvable black-box groups for any group family has canonical generator sets.

Lemma 4.3.3 *Let $\mathcal{B} = \{B_m\}_{m>0}$ be a group family such that $|B_m| \leq 2^{q(m)}$ for a polynomial q. Let $G < B_m$ be a finite solvable group and $G = G_0 > G_1 > \ldots > G_{k-1} > G_k = e$ be the commutator series of G. Let $T_i = \{h_{i1}, h_{i2}, \ldots, h_{is_i}\}$ be a set of distinct coset representatives corresponding to an independent set of generators for the abelian group $H_i = G_{i-1}/G_i$. Then for any i, $1 \leq i \leq k$, the ordered set[1] $S_i = \bigcup_{j=i}^{k} T_j$ forms a canonical generator set for the group G_i and $|S_i| \leq q(m)$. Thus the class of solvable groups from $\mathcal{B}$ has canonical generator sets.*

Proof. Let $\mathcal{B} = \{B_m\}_{m>0}$ be a group family such that $|B_m| \leq 2^{q(m)}$ for a polynomial q. Let $G < B_m$ be a finite solvable group and $G = G_0 > G_1 > \ldots > G_{k-1} > G_k = \langle e \rangle$ be the commutator series of G. Recall that subgroups of solvable groups are solvable. We prove by induction on $k - i$, that for each $i : 1 \leq i \leq k$, $\bigcup_{j=i+1}^{k} T_j$ is a canonical generator set for G_i. For the base case, when $k - i = 1$ it clearly holds, since G_{k-1} is an abelian group and T_k is an independent set of generators for G_{k-1}. Suppose, as induction hypothesis, that $\bigcup_{j=i+1}^{k} T_j$ is a canonical generator set for G_i. Consider

[1]The elements of the set $\bigcup_{j=i}^{k} T_j$ are ordered on increasing values of the index j, and lexicographically within each set T_j.

the group G_{i-1}. The factor group G_{i-1}/G_i is generated by the independent set of generators $X_i = \{h_{i1}G_i, h_{i2}G_i \ldots, h_{ik_i}G_i\}$. An element $g \in G_{i-1}$ belongs to exactly one right coset in G_{i-1}/G_i. Let that right coset be hG_i. Thus $g = hg'$ for some $g' \in G_i$. Now, since X_i is an independent generator set of the abelian group G_{i-1}/G_i, there are unique indices $\{l_{i1}, l_{i2}, \ldots, l_{ik_i}\}$ such that $hG_i = h_{i1}^{l_{i1}} \ldots h_{ik_i}^{l_{ik_i}} G_i$. Consequently, $\{l_{i1}, l_{i2}, \ldots, l_{ik_i}\}$ are unique indices such that $g = hg' = h_{i1}^{l_{i1}} \ldots h_{ik_i}^{l_{ik_i}} g''$, where $g'' \in G_i$. By induction hypothesis, $\bigcup_{j=i+1}^{k} T_j$ is a canonical generator set for G_i, implying that g'' can be uniquely expressed as a product of powers of the elements in $\bigcup_{j=i+1}^{k} T_j$. We have proved the induction step that every $g \in G_{i-1}$ can be uniquely expressed as a product of powers of the elements in $\bigcup_{j=i}^{k} T_j$. Thus, $\bigcup_{j=i}^{k} T_j$ is a canonical generator set for G_{i-1}.

Finally we show that $q(m)$ bounds $|\bigcup_{i=1}^{k} T_i|$. Observe that since for each i, $1 \leq i \leq k$ the set X_i is an independent generator set for the quotient group G_{i-1}/G_i, it holds that $|T_i| \leq \log(|G_{i-1}/G_i|)$. Therefore, it follows that $|\bigcup_{i=1}^{k} T_i| \leq \sum_{1 \leq i \leq k} \log(|G_{i-1}/G_i|) = \log(|G|) \leq q(m)$ since $|G| \leq 2^{q(m)}$. ■

In the rest of this section, we will consider the problem of constructing a canonical generator set from an arbitrary generator set for a solvable black-box group. Since from Lemma 4.3.3 a canonical generator set for a solvable group is an ordered collection of independent generator sets for the abelian factor groups of the commutator series of the solvable group, in the next subsection, we consider the problem of computing an independent generator set for an abelian factor group.

4.3.1 Constructing an Independent Generator Set for Abelian Factor Groups

Here we will design a deterministic oracle algorithm FACTINDGENERATOR which takes generator sets for two groups G and H such that H is normal in G and G/H is abelian, and computes coset representatives for an independent generator set for G/H. More precisely, we prove the following theorem.

Theorem 4.3.4 *Let $\mathcal{B} = \{B_m\}_{m>0}$ be a group family. Then there exists a deterministic oracle algorithm* FactIndGenerator *such that* FactIndGenerator *takes $\langle m, S, Y\rangle$ as input, where $S, Y \subseteq B_m$, and L_{ig} as oracle. Suppose the input $\langle m, S, Y\rangle$ satisfies the following promise:*

1. *Y is a canonical generator set for the solvable group $\langle Y\rangle$.*
2. *$\langle Y\rangle$ is normal in $\langle S\rangle$ and $\langle S\rangle/\langle Y\rangle$ is abelian.*

Then FactIndGenerator *outputs $|\langle S\rangle/\langle Y\rangle|$ and a set of independent generators for $\langle S\rangle/\langle Y\rangle$. Furthermore,* FactIndGenerator *runs in time polynomial in $|\langle m, S, Y\rangle|$, and makes 1-guarded queries to L_{ig}. The behavior of* FactIndGenerator *is unspecified if the input does not satisfy the promise.*

Notice here that FactIndGenerator is using as oracle the same NP language L_{ig} defined in the previous section.

Let us recall the definition of the language L_2' which is one of the main components of L_{ig}.

$$L_2' \triangleq \{\langle m, S, g\rangle \mid g \in B_m, S \subseteq B_m, \text{ and } \forall h \in S; \exists l_h \leq o(h) \text{ such that } g = \Pi_{h\in S} h^{l_h}\}$$

Now we make an important observation that for an instance $\langle m, S, g\rangle$ of L_2', if S is a canonical generator set of a solvable group $\langle S\rangle$ then $\langle m, S, g\rangle \in L_2'$ if and only if $g \in \langle S\rangle$. That is, if a solvable group is presented by a canonical generator set, then Membership Testing can be done using the language L_2'. Moreover, the machine M_2' (given in Proposition 4.2.11) will have at most one accepting path on such instances. We state these observations as a proposition.

Proposition 4.3.5 *Let $\langle m, S, g\rangle$ be an instance of L_2' such that S is a canonical generator for the group $\langle S\rangle$. Then $\langle m, S, g\rangle \in L_2'$ if and only if $g \in \langle S\rangle$. Moreover, for those inputs $\langle m, S, g\rangle \in L_2'$ such that S is canonical generator set for the group $\langle S\rangle$, M_2' will have a unique accepting path.*

Because of the above property of L_2', we have lemmas identical to Lemmas 4.2.12 and 4.2.13 for computing order and exponents over solvable groups also. We state them here.

Lemma 4.3.6 *Let $\mathcal{B} = \{B_m\}_{m>0}$ be a group family. Then there exists a deterministic polynomial-time oracle algorithm* EXPONENT *which takes $\langle m, S, X, g\rangle$ as input and L_2 as oracle such that, if the input satisfies the promise that S is a canonical generator set for the solvable group $\langle S\rangle$, $X \subseteq S$ and $g \in \langle S\rangle$, then* EXPONENT *outputs the unique X-exponent of g. Furthermore* EXPONENT *makes only 1-guarded queries to the oracle. The behavior of* EXPONENT *is unspecified if the input does not satisfy the promise.*

Lemma 4.3.7 *Let $\mathcal{B} = \{B_m\}_{m>0}$ be a group family. Then there exists a deterministic polynomial-time oracle algorithm* ORDER *which takes as input $\langle m, g, X, Y\rangle$, and $0L_1 \cup 1L_2'$ as oracle such that, if the input satisfies the promise that $X, Y \subseteq B_m$, Y is a normal subgroup of X, $g \in \langle X\rangle$ and Y is a canonical generator set for $\langle Y\rangle$, then* ORDER *outputs $o(g\langle Y\rangle)$. Furthermore* ORDER *makes only 1-guarded queries to the oracle. The behavior of* ORDER *is unspecified if the input does not satisfy the promise.*

Finally we come to the design of FACTINDGENERATOR. The design of FACTIND-GENERATOR follows very closely that of INDEPENDENTGENERATOR. Observe that the Lemma 4.2.6, 4.2.7 and Proposition 4.2.14, which are important in the design of INDEPENDENTGENERATOR, are purely group-theoretic and hold for abelian factor groups also.

Before giving a formal description of FACTINDGENERATOR, we shall briefly explain how the algorithm works.

Let $\mathcal{B}$ be a group family. Let $G < B_m$ be a group and H be a normal solvable subgroup of G such that the factor group G/H is abelian. We are interested in computing an independent generator set for G/H. Let the group G/H be presented by a set X of coset representatives of generators of G/H and a canonical generator set Y for H. It is clear that if prime factorization of $|B_m|$ is known, then using method given by Lemma 4.2.7 we can compute a set of coset representatives for a generator set of the Sylow subgroups of G/H. So we shall assume that G/H is an abelian p-group for prime p. We use the method given by the Lemma 4.2.6 for converting X into a set of representatives for independent generator set for G/H.

Let G/H be of type $(m_1, \ldots, m_s)$ and $g_1H, \ldots, g_iH$ be a set of independent elements of G/H of orders $p^{m_1}, \ldots, p^{m_i}$ respectively. Firstly we have to find an element $g'_{i+1} \in X$ such that $g'_{i+1}H$ will be of order $p^{m_{i+1}}$ in the group $\langle \{g_1H, \ldots, g_iH\} \rangle$. Then for getting an element $g_{i+1}H$ independent of $g_1H, \ldots, g_iH$ of order $p^{m_{i+1}}$, we have to compute the $\{g_1H, \ldots, g_iH\}$-exponent of $(g'_{i+1}H)^{p^{m_{i+1}}}$.

Notice that, we have to do order and exponent computations over factor groups of G/H which itself is a factor group. Next we state an isomorphism theorem by which we can avoid computations over factor groups of factor groups. The theorem is elementary and can be seen in any standard text book in group theory [Hal59, Bur55].

Theorem 4.3.8 (Isomorphism Theorem) *Let H and K be normal subgroups of G and $H < K$. Then K/H is normal in G/H and the map $gH(K/H) \to gK$ from the factor group $(G/H)/(K/H)$ to G/K is an isomorphism of $(G/H)/(K/H)$ to G/K.*

Next we state a proposition which will help us in using the algorithm Order for order computations. The proof is implicit in the proof of Lemma 4.3.3 and is omitted here.

Proposition 4.3.9 *Let H, G be solvable and H be a normal subgroup of G such that G/H is abelian. Let $X \subseteq G$ be a set of coset representatives of an independent* *generator set for G/H and Y be a canonical generator set for H. Then the ordered set $S = X \cup Y$ is a canonical generator set for G, where elements from X are before those from Y in S.*

From the above two results, it follows that, the order of any element $gH \in G/H$ in the group $\langle\{g_1H, \ldots, g_iH\}\rangle$ is same as the order of $g \in G$ in the group $\langle\{g_1, \ldots, g_i\} \cup Y\rangle$. So the algorithm Order can be used to compute this. Moreover, since $\{g_1, \ldots, g_i\} \cup Y$ is a canonical generator set, the Order makes only 1-*guarded* queries to L_2'.

Now we see how we can use the algorithm Exponent to compute exponents.

Proposition 4.3.10 *Let G be a solvable group and $H < G$ be a normal subgroup of G such that G/H is abelian and Y be a canonical generator set for H. Let $\{g_1H, \ldots, g_iH\}$ be a set of independent elements of G/H. Then for any $gH \in \langle\{g_1H, \ldots, g_iH\}\rangle$, the $\{g_1H, \ldots, g_iH\}$-exponent of gH is same as the $\{g_1, \ldots, g_i\}$-exponent of g with respect to the canonical generator set $\{g_1, \ldots, g_i\} \cup Y$.*

Proof. Since $\{g_1H, \ldots, g_iH\}$ are independent, it follows that the $\{g_1H, \ldots, g_iH\}$-exponent of gH is unique. Let it be $(l_1, \ldots, l_i)$. Then $gH = (g_1H)^{l_1} \ldots (g_iH)^{l_i}$. This means that there is a unique $x \in H$ such that $g = g_1^{l_1} \ldots g_i^{l_i} x$. The proposition follows. ■

It follows that the algorithm Exponent can be used appropriately for the computation of exponents over a set of independent elements of a abelian factor group.

The rest of the computation is exactly same as that in IndependentGenerator. Finally we give the description of algorithm FactIndGenerator. The proof that the algorithm behaves as given in Theorem 4.3.4 is very similar to that of Theorem 4.2.5.

```
FactIndGenerator(m, S, Y)
n ← |B_m|;
p_1^{e_1} p_2^{e_2} ... p_r^{e_r} ← Factor(n);
for i : 1 ≤ i ≤ r
do Compute a set of coset representatives for the generator
    set X_i for the p_i-Sylow subgroup of G/H;
    /* Using Lemma 4.2.7 */
end-for
for i : 1 ≤ i ≤ r
do S_i ← φ; N_i ← 1; T_i ← ()
    /* S_i stands for the representatives for a set of independent
    generators for p_i-Sylow subgroup of G/H so far constructed */
    while ∃ h ∈ X_i such that ⟨m, S_i ∪ Y, h⟩ ∉ L'_2
    do  Find an element h_j ∈ X_i such that
        p_i^{l_j} ←Order(m, h_j, X_i, S_i ∪ Y) is maximum;
        (x_g | g ∈ S_i) ← Exponent(m, S_i, Y, h_j);
        S_i ← S_i ∪ {h_j Π_{g∈S_i} g^{-x_g/p_i^{l_j}}};
        N_i ← p_i^{l_j}.N_i; T_i ← T_i ∪ (l_j)
    end-while
end-for
Output ∪_{i=1}^r S_i as representatives for an independent generator set for G/H;
Output Π_{1≤i≤r} N_i as the order of G/H;
Output T_i as type of the p_i-Sylow subgroup of ⟨S⟩/⟨Y⟩ for 1 ≤ i ≤ r.
```

4.3.2 Constructing a Canonical Generator Set

Now we are ready to design a deterministic oracle algorithm (we shall call it CanonicalGenerator) which takes an arbitrary generator set of a solvable group G and converts it into a canonical generator set for G. The algorithm makes queries to the language L_{ig} defined earlier in this chapter. Again, for applications to our upper bound proofs, we will make sure that CanonicalGenerator makes only 1-guarded queries to L_{ig}. We state the behavior of CanonicalGenerator as a theorem. This subsection is devoted to the proof of this theorem.

Theorem 4.3.11 *Let $\mathcal{B} = \{B_m\}_{m\geq 0}$ be a group family. Then there is a deterministic oracle algorithm* CanonicalGenerator *such that* CanonicalGenerator *takes $\langle m, S\rangle$ as input and L_{ig} as oracle, and outputs a canonical generator set for $\langle S\rangle$ if $\langle S\rangle$ is solvable and outputs* NOT SOLVABLE *otherwise. Moreover,* CANONICALGENERATOR *runs in time polynomial in the length of the input and makes only 1-guarded queries to L_{ig}.*

Before going into the formal proof of the theorem, we give the basic ideas behind the proof. Firstly we shall see the usefulness of the algorithm FACTINDGENERATOR (Theorem 4.3.4) in the design of CANONICALGENERATOR. Let S be an arbitrary generator set for a solvable group $G < B_m$. Let $G = G_0 > \ldots > G_i > \ldots > G_k = \{e\}$ be the commutator series of G. (Notice that $k \leq p(m)$ where p is the polynomial bounding the length of the encoding of elements of the group in the group family. This follows from the fact that $|B_m| \leq 2^{p(m)}$ and Lagrange's theorem.) From Lemma 4.3.3, a canonical generator set for a solvable group is an ordered collection of independent generator sets for each abelian factor groups G_{i-1}/G_i. So, suppose we have given a set of generators S_i for each of the groups G_i in the commutator series, then we can construct a canonical generator set for G by a series of simulations of FACTINDGENERATOR on inputs $\langle m, S_{i-1}, S_i'\rangle$, for $k \geq i \geq 1$, starting from $i = k$ and decrementing i by one after each simulation. Here S_i' is the output of FACTINDGENERATOR on the previous simulation. So we have the following theorem.

Theorem 4.3.12 *Let $\mathcal{B} = \{B_m\}_{m\geq 0}$ be a group family. Then there is a deterministic oracle algorithm* CANONIZE *such that* CANONIZE *takes $\langle m, S_0, \ldots, S_k\rangle$, $S_i \subseteq B_m$ as input and L_{ig} as oracle. Suppose the input satisfies the promise that $\langle S_0\rangle$ is solvable and for all $0 \leq i \leq k$, S_i generates the i^{th} commutator subgroup of $\langle S_0\rangle$. Then* CANONIZE *outputs canonical generator sets for $\langle S_i\rangle$ for $0 \leq i \leq k$. Moreover,* CANONIZE *runs in time polynomial in the length of the input and makes only 1-guarded queries to L_{ig}. The behavior of* CANONIZE *is unspecified if the input does not satisfy the promise.*

In view of the above theorem, it is clear that the problem of computing a canonical generator set for any solvable group essentially boils down to the problem of computing a short generator set for the commutator subgroup of the group. The following proposition provides a method for this computation. A proof of this is given in the appendix.

Proposition 4.3.13 *Let G be a finite group generated by the set S. Then, the commutator subgroup of G is the normal closure of the set $\{ghg^{-1}h^{-1} \mid g, h \in S\}$ in G.*

The above theorem gives us the following easy polynomial-time oracle algorithm CommutatorSubgroup which takes $\langle m, S\rangle$ as input and Membership Testing as oracle and computes a generator set for the commutator group of $\langle S\rangle$.

CommutatorSubgroup(m, S)
$X \leftarrow \{ghg^{-1}h^{-1} \mid g, h \in S\}$;
while $\exists g \in S;\ x \in X$ such that $\langle m, X, gxg^{-1}\rangle \notin$ Membership Testing
do $X \leftarrow X \cup gxg^{-1}$;
end-while
Output X.

It easily follows from Proposition 4.3.13 that the algorithm CommutatorSubgroup, on input $\langle m, S\rangle$ outputs a generator set for the commutator subgroup $\langle S\rangle'$. Let X_i be the set X at the beginning of the i^{th} iteration of the **while**-loop. If, after the i^{th} iteration, no new element is added to X_i, then X_i is output. Otherwise, if $X_{i+1} = X_i \cup \{g\}$, it follows from Lagrange's theorem that $|X_{i+1}| \geq 2|X_i|$. Hence the number of iterations of the **while**-loop is bounded by $p(m)$, the polynomial bounding the length of the encoding of the elements of the group in the group family.

A straightforward adaptation of the above algorithm for computing a generator set for the commutator group seems difficult because of the queries to Membership

Testing oracle. Suppose we can make sure that whenever a query $\langle m, X, g\rangle$ to Membership Testing is done, X is a canonical generator set for the solvable group $\langle X\rangle$, then we can replace Membership Testing oracle by L_2' and from Proposition 4.3.5 it will follow that query y will be 1-guarded. We make sure this promise, by constructing the commutator series incrementally.

The algorithm CanonicalGenerator works in stages. Let S_i^j denote the partial generator set for the i^{th} element in the commutator series of G_0 constructed at the end of *stage* $(j-1)$. At *stage 1* we have $S_0^1 = S$ and $S_i^1 = \{e\}$ for $1 \leq i \leq p(m)$, where p is the polynomial bounding the length of any element in the group family. Input to *Stage* j is the tuple $\langle i, S_i^j, \ldots, S_{p(m)}^j\rangle$ such that for $l > i$, S_l^j is a canonical generator set for the solvable group $\langle S_l^j\rangle$. During the computation of this stage, we will make sure that for all the queries made to L_{ig}, the groups involved in the queries will be only those for which a canonical generator set has been constructed, thereby making these queries 1-guarded. At the end of the stage, we update each S_i^j to S_i^{j+1} such that S_i^{j+1} is still a subgroup of G_i, the i^{th} commutator subgroup of G_0. To keep the running time within polynomial bound, we make sure that after $p(m)$ stages, there exists k, such that the k^{th} partial commutator subgroup doubles in size. Then from Lagrange's theorem, it will follow that the commutator series will be generated after $p(m)^3$ stages. We now formally prove the theorem.

Proof. (*of Theorem 4.3.11*) We first give a formal description of the algorithm CanonicalGenerator and then prove the correctness.

CanonicalGenerator uses oracle algorithms CheckCommutator and Canonize as subroutines. CheckCommutator takes as input $\langle m, X, Y\rangle$ such that $X, Y \subseteq B_m$ and checks whether $\langle Y\rangle$ contains the commutator subgroup of $\langle X\rangle$. This is done by first checking whether the commutators of all the elements in X are in $\langle Y\rangle$. If this is not the case, the algorithm returns such a commutator. Otherwise, it further checks whether $\langle Y\rangle$ is normal in $\langle X\rangle$. Notice that, to do this it is enough to verify that $\forall x \in X; y \in Y$, $xyx^{-1} \in \langle Y\rangle$. If this condition is false, the algorithm returns an element $xyx^{-1} \notin \langle Y \rangle$. If both the conditions are true, it follows from Proposition 4.3.13 that $\langle Y \rangle$ contains the commutator subgroup of $\langle X \rangle$.

CHECKCOMMUTATOR makes oracle queries to the language L'_2. (weak Membership Testing, defined in the section 4.2.1) for testing membership in $\langle Y \rangle$. It should be noted that, for CHECKCOMMUTATOR to work as intended, Y should be a canonical generator set for the group $\langle Y \rangle$. We will make sure that CANONICALGENERATOR makes calls to CHECKCOMMUTATOR with $\langle m, X, Y \rangle$ as input, only when Y is a canonical generator set for the solvable group $\langle Y \rangle$. A formal description of the subroutine CHECKCOMMUTATOR is given below.

```
CHECKCOMMUTATOR(m, X, Y)
if ∃x_1, x_2 ∈ X, such that ⟨m, Y, x_1 x_2 x_1^{-1} x_2^{-1}⟩ ∉ L'_2
   then g ← x y x^{-1} y^{-1};
        Return g.
   else if ∃x ∈ X, y ∈ Y such that ⟨0^m, Y, x y x^{-1}⟩ ∉ L'_2
           then g ← x y x^{-1};
                Return g.
           else g ← YES;
                Return g.
        end-if
end-if
```

The subroutine CANONIZE is the algorithm promised by Theorem 4.3.12 for computing a canonical generator set for a solvable black-box group G, given an arbitrary generator set for the commutator series of G. We use the notation $[\text{CANONIZE}()]_l$ to denote the generator set produced by CANONIZE for the l^{th} element G_l, in the commutator series of G.

Following is the description of the algorithm CANONICALGENERATOR. CANONICALGENERATOR makes queries L_{ig} through the subroutines CANONIZE and CHECKCOMMUTATOR.

CanonicalGenerator(m, S)

1 **Stage** 0
2 $S_1^0 \leftarrow S$; $S_i^0 \leftarrow \{e\}$ for $1 \leq i \leq p(m)$;
3 $i \leftarrow 1$;
4 **Stage** j (Input to this stage is $\langle i, S_i^j, \ldots, S_{p(m)}^j \rangle$)
5 $k \leftarrow i$;
6 $g \leftarrow$ CheckCommutator(m, S_k^j, S_{k+1}^j);
7 **while** $g \neq$ YES
8 **do** $S_{k+1}^j \leftarrow S_k^j \cup \{g\}$;
9   $k \leftarrow k+1$;
10   **if** $k = p(m)$
11   **then Output** NOT SOLVABLE.
12   **end-if**
13   $g \leftarrow$ CheckCommutator(m, S_k^j, S_{k+1}^j);
14 **end-while**
15 **if** $k = 1$
16   **then Output** [Canonize$(S_1^j, S_2^j, \ldots, S_{p(m)}^j)]_1$.
17   **else** $S_l^{j+1} \leftarrow S_l^j$ for $1 \leq l \leq (k-1)$;
18     $S_l^{j+1} \leftarrow$ [Canonize$(S_k^j, S_{k+1}^j, \ldots, S_{p(m)}^j)]_l$ for $k \leq l \leq p(m)$;
19     $i \leftarrow (k-1)$;
20   goto **Stage** $j+1$;
21 **end-if**

Now we are ready to prove the correctness of CanonicalGenerator. We first prove a series of claims, from which the correctness will follow easily.

Claim 4.3.13.1 *In the algorithm* CanonicalGenerator, *at any stage j, it holds that $\forall i; 1 \leq i < p(m)$, $\langle S_{i+1}^j \rangle < \langle S_i^j \rangle'$.*

Proof. We prove this by induction on the stages. For the base case, when $j = 0$, it is clear that the claim holds. Assume that it is true for $(j-1)^{\text{th}}$ stage. Now consider S_{i+1}^j and S_i^j. Depending on how the sets S_{i+1}^j and S_i^j are updated in *lines*-17,18 of CanonicalGenerator, we have the following cases.

Case 1. $S_i^j = S_i^{j-1}$; $S_{i+1}^j = S_{i+1}^{j-1}$. In this case, from the induction hypothesis, it is clear that $\langle S_{i+1}^j \rangle < \langle S_i^j \rangle'$.

Case 2. $S_i^j = S_i^{j-1} \cup \{g_i\}$; $S_{i+1}^j = S_{i+1}^{j-1}$. From the induction hypothesis, it follows that $\langle S^j_{i+1}\rangle = \langle S^{j-1}_{i+1}\rangle < \langle S^{j-1}_i\rangle' < \langle S^j_i\rangle'$.

Case 3. $S^j_i = S^{j-1}_i$; $S^j_{i+1} = S^{j-1}_{i+1} \cup \{g_{i+1}\}$. The element g_{i+1} is added to the set S^{j-1}_{i+1} at *line*-8 of the algorithm, where g_{i+1} is the element returned by the subroutine CHECKCOMMUTATOR. Suppose g_{i+1} is a commutator of the set $S^j_i = S^{j-1}_i$. Then $g_{i+1} = xyx^{-1}y^{-1}$ for some elements $x, y \in S^j_i$. From induction hypothesis and the definition of the commutator subgroup of a group, it follows that $\langle S^j_{i+1}\rangle = \langle S^{j-1}_{i+1} \cup \{g_{i+1}\}\rangle < \langle S^j_i\rangle'$. On the other hand, suppose g_{i+1} is of the form xyx^{-1} for some $x \in S^j_i = S^{j-1}_i$ and $y \in S^{j-1}_{i+1}$. We have, $\langle S^{j-1}_{i+1}\rangle < \langle S^{j-1}_i\rangle' = \langle S^j_i\rangle'$. But we know that $\langle S^j_i\rangle'$ is normal in $\langle S^j_i\rangle$. So, in particular $g_{i+1} \in \langle S^j_i\rangle'$ and hence $\langle S^j_{i+1}\rangle < \langle S^j_i\rangle'$.

Case 4 . $S^j_i = S^{j-1}_i \cup \{g_i\}$; $S^j_{i+1} = S^{j-1}_{i+1} \cup \{g_{i+1}\}$. From induction hypothesis, we have $\langle S^{j-1}_{i+1}\rangle < \langle S^{j-1}_i\rangle'$. It follows that $\langle S^{j-1}_{i+1}\rangle < \langle S^{j-1}_i \cup \{g_i\}\rangle$. Now using a very similar argument as in **Case 3**, it is easy to show that $\langle S^j_{i+1}\rangle < \langle S^j_i\rangle'$.

Hence the claim. ■

Claim 4.3.13.2 *In* CANONICALGENERATOR, *the input* $\langle i, S^j_i, S^j_{i+1}, \ldots, S^j_{p(m)}\rangle$ *to any stage* j*, is such that for all* $i < t \leq p(m)$*,* S^j_t *is a canonical generator for the solvable group* $\langle S^j_t\rangle$.

Proof. We prove this by induction. For $j = 1$, it is easily verified that the claim is true. Let us assume that the claim is true for j^{th} stage. Let $\langle i, S^j_i, \ldots, S^j_{p(m)}\rangle$ be the input to the j^{th} stage. Let the **while**-loop is exited through *line*-14 after l iterations with the value of g='YES'. (If the loop is exited through *line*-11, then there are no more stages to be considered). Then the value of $k = i + l$ and for $t > k$, S^j_t is not updated inside the loop, and hence by induction hypothesis, it remains a canonical generator set for the solvable group $\langle S^j_t\rangle$. Since the value of $g = $ CHECKCOMMUTATOR(m, S^j_k, S^j_{k+1}) is 'YES' we have that $\langle S^j_k\rangle' < \langle S^j_{k+1}\rangle$. From Claim 4.3.13.1 we have $\langle S^j_{k+1}\rangle < \langle S^j_k\rangle'$. Hence $\langle S^j_{k+1}\rangle = \langle S^j_k\rangle'$. It follows that $\langle S^j_k\rangle$ is solvable and S^j_t for $k \leq t \leq p(m)$ are generator sets for the commutator series of $\langle S^j_k\rangle$. Hence at *line*-18, CANONIZE will output a canonical generator set for each of the elements in the commutator series of $\langle S^j_k\rangle$. At *line*-19, i is updated to $k-1$, and the input to the $j+1^{\text{th}}$ stage $\langle k-1, S_{k-1}^{j+1}, S_k^{j+1}, \ldots, S_{p(m)}^{j+1}\rangle$ where S_t^{j+1} is a canonical generator set for the solvable group $\langle S_t^{j+1}\rangle$ for $k \le t \le p(m)$. Hence the claim. ∎

Claim 4.3.13.3 *In the algorithm* CanonicalGenerator*, for any stage j, it holds that $\exists i$ such that $|\langle S_i^{j+p(m)}\rangle| \ge 2|\langle S_i^j\rangle|$ if stage $j+p(m)$ exists.*

Proof. Notice that, if the algorithm at stage j enters the **while**-loop, then there $\exists i$ such that $S_i^{j+1} = S_i^j \cup g$ for a $g \notin \langle S_i^j\rangle$. So, it is enough to show that the **while**-loop is entered at least once after every $p(m)$ sages, if such a stage exists. Suppose the stage j is entered with the value of $i = i'$. It is clear from the algorithm that if the algorithm never enters the **while**-loop in the next $p(m)$ stages, at stage $(j + p(m) + 1)$, the value of $i = i' - (p(m) + 1) < 0$, for all $i' \le p(m)$, which is impossible, since the algorithm is terminated when the value of $i = 0$. Hence the claim. ∎

To complete the proof of the theorem, first we shall see that the algorithm CanonicalGenerator runs in time polynomial in the length of the input. Observe that it is enough to show that the number of stages executed by the algorithm is bounded by a polynomial, since the number of iterations of the **while**-loop in *lines* 7-14 is bounded by $p(m)$. Now, the claim is that the the number of stages executed by the algorithm is bounded by $2p^3(m)$. Firstly, notice that for any $H < B_m$, $|H| \le 2^{p(m)}$. Hence for any j, $\prod_{i=1}^{p(m)} |\langle S_i^j\rangle| \le 2^{p^2(m)}$. Suppose the claim is false. Now from Claim 4.3.13.3 it follows that, $\prod_{i=1}^{p(m)} |\langle S_i^{j+p(m)}\rangle| \ge 2\prod_{i=1}^{p(m)} |\langle S_i^j\rangle|$. Hence $\prod_{i=1}^{p(m)} |\langle S_i^{2p^3(m)}\rangle| > 2^{p^2(m)}$, a contradiction.

Now we shall see that CanonicalGenerator makes only 1-guarded queries to L_{ig}. Let us first see that the queries to L_{ig} through CheckCommutator are 1-guarded. It is enough to show that whenever CanonicalGenerator calls CheckCommutator with argument $\langle m, S_k^j, S_{k+1}^j\rangle$ in *stage* j, S_{k+1}^j is a canonical generator set. But from Claim 4.3.13.2, the input $\langle i, S_i^j, S_{i+1}^j, \ldots, S_{p(m)}^j\rangle$ to any stage j, is such that for all $i < t \leq p(m)$, S_t^j is a canonical generator for the solvable group $\langle S_t^j \rangle$. Now, by inspecting the description of the algorithm, it follows that whenever CANONICALGENERATOR calls CHECKCOMMUTATOR with argument $\langle m, S_k^j, S_{k+1}^j \rangle$, S_{k+1}^j is a canonical generator set.

To see that the queries to L_{ig} through CANONIZE are 1-guarded, notice that calls to CANONIZE are made outside the **while**-loop. This means that CHECK-COMMUTATOR with input (m, S_k^j, S_{k+1}^j) returns YES. That is $\langle S_k^j \rangle' < \langle S_{k+1}^j \rangle$. Hence $\langle S_k^j \rangle' = \langle S_{k+1}^j \rangle$ from Claim 4.3.13.1. So it follows that calls to CANONIZE with argument $\langle S_i^j, S_{i+1}^j, \ldots, S_{p(m)}^j \rangle$ will be such that S_l^j for $i \leq l \leq p(m)$ will generate the commutator series of S_i^j for all i. It follows from Theorem 4.3.12 that queries to L_2' will be 1-guarded.

Finally, we show that the above algorithm on input (m, S), outputs a canonical generator set for the group $G = \langle S \rangle$ if G is solvable and outputs NOT SOLVABLE otherwise. Now, observe that if $H_1 < H_2$ are two finite groups, $H_1' < H_2'$. Hence it follows from Claim 4.3.13.1 that, $\langle S_i^j \rangle < G_i$ for any i at any stage j, where G_i is the i^{th} element in the commutator series of G. We know that after the execution of $2p^3(m)$ stages, the algorithm outputs either a set $X \subseteq B_m$ or NOT SOLVABLE. Suppose it outputs NOT SOLVABLE in stage j. This happens after the value of the variable inside the **while**-loop is assigned $p(m)$. From the description of the algorithm inside the loop, it follows that the group $\langle S_{p(m)}^j \rangle$ does not contain the commutator subgroup of $\langle S_{p(m)-1}^j \rangle$. But if G where solvable, then we know that $G_{p(m)} = \{e\}$ and since $\langle S_{p(m)}^j \rangle < G_{p(m)}$ from Claim 4.3.13.1, we have a contradiction.

Suppose the algorithm outputs a set $X \subseteq B_m$ at *line*-16 in stage j. Thus the value of the variable k is 1. Notice that, inside the **while**-loop, the value of k is only incremented. This implies that at stage j the **while**-loop is not entered (the value of i could not have become 0 at a previous stage). So input to stage j is $\langle 1, S_1^j, \ldots, S_{p(m)} \rangle$. From Claim 4.3.13.2, it follows that for all $2 \leq t \leq p(m)$, $\langle S_t^j \rangle$ is solvable and S_t^j is a canonical generator set for the group $\langle S_t^j \rangle$. From the value of g =YES and Claim 4.3.13.1, it follows that $\langle S_1^j \rangle' = \langle S_2^j \rangle$. Also, since $S_1^j = S$ for any stage j, it follows that S_i^j generates the i^{th} element in the commutator series of $\langle S \rangle = G$. Hence, from Theorem 4.3.12, it follows that $[\textsc{Canonize}(S_1^j, \ldots, S_{p(m)}^j)]_1$ is a canonical generator set for G. This completes the proof of the the theorem.

∎

4.4 Solvable Group Problems are in SPP

Finally, in this section, we show that all the three basic problems over solvable groups we consider here are in SPP. Notice that, the problem of checking whether a black-box group is solvable or not is in the class SPP follows from Theorem 4.3.11 and Lemma 2.0.2. Now we show how the algorithm CanonicalGenerator can be modified to get the required upper bound.

Theorem 4.4.1 *Over any group family $\mathcal{B}$, the problems* Membership Testing, Order Verification, Group Isomorphism *over the subclass of solvable groups are in* SPP *and hence low for the classes* PP*,* $\mathrm{C_=P}$*and* $\mathrm{Mod}_k\mathrm{P}$ *for* $k \geq 2$*.*

Proof. The proof is very similar to the upper bound proofs for abelian groups given in Theorem 4.2.16. Consider Membership Testing. Let $\mathcal{B}$ be a group family. Consider an oracle NP machine M which on input $\langle m, S, g \rangle$ converts the set S to a canonical generator set if S generates a solvable group, by simulating the algorithm CanonicalGenerator on input $\langle m, S \rangle$. Let S' be the canonical generator set thus constructed. Then M makes one query $\langle m, S', g \rangle$ to the language L_2' and it accepts if the answer is 'YES'. If the answer to the query is 'NO' then M branches into an accepting and a rejecting path, thus producing a zero *gap*. So, *gap* produced by M is 0 if $g \notin \langle S' \rangle = \langle S \rangle$ and *gap* is 1 if $g \in \langle S' \rangle = \langle S \rangle$. Since S' is a canonical generator set, It follows that M makes only 1-*guarded* queries to L_{ig}. Also, M

runs in time polynomial in the input size. Hence from Lemma 2.0.2, it follows that Membership Testing$\in$ SPP.

To show that Order Verification is in SPP, notice that the algorithm CANONICALGENERATOR can be easily modified to get an algorithm M_{ov} so that on input $\langle m, S\rangle$, it computes $|\langle S\rangle|$, by making only 1-guarded queries to L_{ig}. Hence, again by Lemma 2.0.2, it follows that Order Verification is in SPP.

Finally we show that Group Isomorphism for solvable groups is in SPP. Let $\langle m, S_1, S_2\rangle$ be an instance of Group Isomorphism. Let Let $\langle S_1\rangle = G_0 > G_1 > \ldots > G_k = \langle e\rangle$ and $\langle S_2\rangle = H_0 > H_1 > \ldots > H_k = \langle e\rangle$ be the two commutator series. $\langle S_1\rangle$ and $\langle S_2\rangle$ are isomorphic iff for each $i : 1 \leq i \leq k$, the abelian factor groups G_{i-1}/G_i and H_{i-1}/H_i are isomorphic. G_{i-1}/G_i and H_{i-1}/H_i are isomorphic if the types of G_{i-1}/G_i and H_{i-1}/H_i are the same. Now, the algorithm CANONICALGENERATOR can be modified to get a polynomial-time oracle Turing machine M_{is} which computes the type of each of the abelian factor groups involved in the commutator series of the solvable groups given in the input, making only 1-guarded queries to L_{ig} and producing a *gap*=1 if $\langle S_1\rangle$ is isomorphic to $\langle S_2\rangle$ and a *gap*=0 otherwise. It follows that Group Isomorphism is in SPP. ■

4.5 Summary

In this chapter we considered the complexity of three basic computational problems over solvable black-box groups namely Membership Testing, Order Verification and Isomorphism Testing. These problems are neither known to be in P nor known to be hard for NP. We showed that these problems are in the low complexity counting class SPP. Hence these problems are low for many counting classes and unlikely to be hard for NP. The upper bound is built upon a constructive version of the fundamental theorem of finite abelian groups. For extending the result from abelian groups to solvable groups, we extend the concept of an independent generator set of an abelian group to a similar, special kind of generator set (we call it canonical generator set) for solvable groups.

This concludes the first part of the thesis. In the next chapter, we will study the problem of computing a generator set for a group if we are provided with a membership testing oracle. We pose this problem in learning-theoretic framework and study its complexity in the *teaching assistant* model of learning.