Theoretical Computer Science 242 (2000) 199–218

Theoretical Computer Science

www.elsevier.com/locate/tcs

[illegible]

V. Arvind *, N.V. Vinodchandran

Department of Computer Science, Institute of Mathematical Sciences, CIT Campus, Madras 600113, India

Received July 1995; revised July 1997
Communicated by M. Nivat

Abstract

A *group family* is a countable family $\mathscr{B} = \{B_n\}_{n>0}$ of finite *black-box groups*, i.e., the elements of each group B_n are uniquely encoded as strings of uniform length (polynomial in n) and for each B_n the group operations are computable in time polynomial in n. In this paper we study the complexity of NP sets A which has the following property: the set of solutions for every $x \in A$ is a subgroup (or is the right coset of a subgroup) of a group $B_{i(|x|)}$ from a given group family $\mathscr{B}$, where i is a polynomial. Such an NP set A is said to be defined over the group family $\mathscr{B}$.

Decision problems like Graph Automorphism, Graph Isomorphism, Group Intersection, Coset Intersection, and Group Factorization for permutation groups give natural examples of such NP sets defined over the group family of all permutation groups. We show that any such NP set defined over *permutation groups* is low for PP and $\mathrm{C_{=}P}$.

As one of our main results we prove that NP sets defined over *abelian black-box groups* are low for PP. The proof of this result is based on the decomposition theorem for finite abelian groups. As an interesting consequence of this result we obtain new lowness results: Membership Testing, Group Intersection, Group Factorization, and some other problems for abelian black-box groups are low for PP and $\mathrm{C_{=}P}$.

As regards the corresponding counting problem for NP sets over any group family of arbitrary black-box groups, we prove that exact counting of number of solutions is in $\mathrm{FP^{AM}}$. Consequently, none of these counting problems can be #P-complete unless PH collapses. © 2000 Elsevier Science B.V. All rights reserved.

Keywords: Group Family; Group definable languages; NP-complete

1. Introduction

Graph Isomorphism [1] (GI) is an algorithmic problem of intriguing complexity. To date no polynomial-time algorithm has been found for it. On the other hand, there

* Corresponding author. E-mail: arvind,vinod@imsc.ernet.in.

[1] The problem of deciding whether two given labeled graphs are isomorphic.

is strong evidence that GI is not NP-complete: in [7] it was shown that GI is in NP $\cap$ co-AM, which implies that GI cannot be NP-complete unless the polynomial hierarchy collapses to Σ_2^p [7, 17]. More recently, it was shown in [15] that GI is *low* for PP. In other words, GI is powerless as oracle for PP computations (where PP is the language class corresponding to #P).

Similar results also hold for several group-theoretic problems. In [4] it is shown that Group Intersection and the Coset Intersection problems for permutation groups are in NP $\cap$ co-AM. In [15] it is shown for permutation groups that the Group Intersection problem, Group Factorization, Coset Intersection and Double Coset Membership (formal definitions are given later) are all low for PP.

The underlying group-theoretic structure of these problems plays a crucial role in the proofs of the above-mentioned complexity results. For example, in the case of Graph Isomorphism the proofs of the results rely crucially on the following: (a) the set of automorphisms of a graph on n vertices is a subgroup of S_n, the symmetric group on n elements, and (b) if G_1 and G_2 are two isomorphic graphs and $Aut(G_1)$ denotes the automorphism group of G_1 then the set of all isomorphisms from G_1 to G_2 is a right coset $Aut(G_1)\psi$ of $Aut(G_1)$, where ψ is an isomorphism from G_1 to G_2. Similarly, for the group-theoretic problems mentioned above, the derivation of the lowness results is based on the inherent group-theoretic structure of the problems.

The object of our study is to look for similarity in the inherent structure of the problems mentioned above, in order to explain why they have very similar structural complexity. Intuitively, it appears unlikely that the instances of an NP-complete language can have solutions sets that enjoy a nice algebraic structure (which the above problems have). This motivates us to study the following class of NP languages A defined as follows: for each $x \in A$, the set of witnesses for x, w.r.t. some NP machine, encodes a right coset of some finite group. In this paper we mainly investigate the lowness of these languages for the class PP.

Before we explain our results, we give some definitions. We first define the notion of *group families*, which were introduced in a somewhat different context in [2].

Definition 1.1 (*Babai* [2]). A *group family* is a countable sequence $\mathscr{B} = \{B_m\}_{m\geqslant 1}$ of finite groups B_m, such that there are polynomials p and q satisfying the following conditions. For each $m\geqslant 1$, elements of B_m are uniquely encoded as strings in $\Sigma^{p(m)}$. The group operations (inverse, product and testing for identity) of B_m can be performed in time bounded by $q(m)$, for every $m\geqslant 1$. The order of B_m is computable in time bounded by $q(m)$, for each m. We refer to the groups B_m of a group family and their subgroups as *black-box* groups.[2]

We give two examples. Let S_n denote the permutation group on n elements. Then, SYM $= \{S_n\}_{n\geqslant 1}$ is a group family of all permutation groups S_n. Let $GL_n(q)$ denote the

[2] Note that black-box groups we define above are a restricted version of black-box groups introduced in [2].The black-box group defined in [2] is technically more general. There the black-box group is defined so as to incorporate factor groups.

group of all $n \times n$ invertible matrices over the finite field F_q of size q. The collection $\mathrm{GL(q)} = \{GL_n(q)\}_{n \geqslant 1}$ is a group family.

Next, we make precise our definition of the class of NP languages, the solution sets of whose instances form right cosets of finite groups.

Let A be an NP language defined by a polynomial-time computable relation R. For any $x \in A$ let $Sol_R(x)$ denote the set of witnesses for x w.r.t R.

Definition 1.2. An NP language $A \subseteq \Sigma^*$ is said to be *group-definable* if there is a group family $\mathscr{B} = \{B_m\}_{m \geqslant 1}$ and a polynomial i, and A is defined by a polynomial-time computable relation R, such that for every $x \in A$, $Sol_R(x)$ is a right coset of some subgroup of $B_{i(|x|)}$. More specifically, we say that A is group-definable over $\mathscr{B}$ via relation R.

It is easy to see from the definition that GI and problems Group Factorization, Coset Intersection and Double Coset membership (definitions of these problems are given in Section 4.3) for permutation groups are group-definable over SYM. In [15], it is shown that these problems are in the class LWPP and hence low for PP. However, [15] give different lowness proofs for each of the above-mentioned problems. The next theorem captures these PP-lowness results in a single statement. We omit the proof because it can be proved exactly on same lines as [15].

Theorem 1.3. *Every language that is group-definable over* SYM *is in* LWPP *and hence low for* PP.

This result motivates us to explore the complexity of group-definable languages over arbitrary group families.

In this paper, we look at the complexity of various subclasses of group-definable languages. We show results which essentially indicate that group-definable languages are unlikely to be NP-complete. More precisely, we show that these subclasses of group-definable languages are low for PP. The following are the subclasses of group-definable languages that we are interested in.

Definition 1.4. Let $A \subseteq \Sigma^*$ be a group-definable language over $\mathscr{B}$ via relation R.
1. For a fixed prime p, A is said to be *p-group-definable* if for every $x \in A$, $Sol_R(x)$ is a right coset of a p-group.
2. A is said to be *prime-power group-definable*, if for every $x \in A$, $Sol_R(x)$ is a right coset of some p-subgroup of $B_{i(|x|)}$ where p is a prime.
3. A is said to be *abelian group-definable* if for every $x \in A$, $Sol_R(x)$ is a right coset of some abelian subgroup of $B_{i(|x|)}$

We show that languages that are p-group-definable, prime-power group-definable, and abelian group-definable are all low for PP. The proofs of lowness of p-group-definable and prime-power group-definable for PP are relatively easy. The proof that abelian group-definable languages are low for PP requires the structure theorem of abelian groups. This result about abelian group-definable languages yields as corollaries the PP-lowness results of [1] for various specific group-theoretic problems.

We also show, in Section 5, that the counting problems corresponding to group-definable languages are in $\mathrm{FP}^{\mathrm{AM}}$, and hence unlikely to be #P-complete unless PH collapses to $\mathrm{FP}^{\mathrm{AM}}$. All these results are in support of our intuition that the solution sets of the instances of NP-complete languages are unlikely to have nice algebraic structure.

The layout of the paper is as follows. In Section 3 we prove that the group-definable languages over prime-power groups are low for PP. In Section 4 we prove the main structural theorem. This section is in three parts. The first subsection deals with developing a formula for counting the number of independent generator sets for an abelain group, which we apply in the proof of the main theorem. The main theorem is proved in Section 4.2. In Section 5, we prove upper bounds on the complexity of counting problems corresponding to group-definable languages.

2. Notations and definitions

2.1. Complexity-theoretic preliminaries

We fix the finite alphabet $\Sigma = \{0,1\}$. For any finite set X, $|X|$ denotes the cardinality of X. For an $x \in \Sigma^*$, $|x|$ denotes the length of x. For any oracle A, the language L in NP(A) if there is a relation $R \in \mathrm{P}(A)$ and a polynomial p such that $x \in L$ iff $\exists y \in \Sigma^{p(|x|)}$ and $\langle x, y\rangle \in R$. Corresponding to the relation R, let $Sol_R(x)$ denote the set $\{y \in \Sigma^{p(|x|)} \mid \langle x, y\rangle \in R\}$, for each $x \in \Sigma^*$.

Let $\mathbf{Z}$ denote the set of integers. 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. Let GapP [11] denote the class of gap-definable functions. For each NP machine M let gap_M denote the GapP function defined by it. The corresponding language class PP is defined as follows: A language L is in PP if there is a GapP function f such that: $x \in L$ iff $f(x) > 0$.

The counting classes of interest in this paper are UP, FewP, SPP and LWPP defined using GapP functions [11]. A language L is in UP if there is an NP machine M accepting L such that M has at most one accepting path on any input. A language L is in FewP if there is polynomial p and an NP machine M accepting L such that M has at most $p(|x|)$ accepting for any input. A language L is in SPP if there is an $f \in$ GapP such that: $x \in L$ implies that $f(x) = 1$, and $x \notin L$ implies that $f(x) = 0$. A language L is in 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 is easy to see that UP $\subseteq$ SPP $\subseteq$ LWPP. An important result that we use frequently is that primality checking is in UP [10].

Let P(A), NP(A) and PP(A) denote the classes of languages obtained by the natural relativization w.r.t. oracle A. $A \subseteq \Sigma^*$ is said to be *low* for PP if PP(A) = PP. In [11] it is shown that every language in LWPP is low for PP.

Another complexity class which is of interest to us is the class AM introduced in [2]. For this and other standard complexity theoretic definitions refer to [6].

2.2. Group-theoretic preliminaries

In this subsection, we give some basic group-theoretic definitions and formalize some notation. Details can be found in textbooks [8, 13].

Definition 2.1 (*Burnside* [8], *Hall* [13]). A *group* is a nonempty set G endowed with a binary operation $*$ such that 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 composition $x * y$ by xy. In this paper we deal with only finite groups.

For any finite groups G, $|G|$ denotes the *order* of G. For an element $g \in G$, the *order* of g (denoted $\mathrm{o}(g)$) is the smallest positive integer such that $g^{\mathrm{o}(g)} = e$, where e is the identity of G. A subset H of G is called a *subgroup* of G (denoted $H < G$) if H is a group under the group operation of G.

A group G is *abelian* if $\forall g_1, g_2, \in G$: $g_1 g_2 = g_2 g_1$. Let S be a subset of a group G. The smallest subgroup of G containing S is called the group *generated by* S and is denoted $\langle S \rangle$. A subset S of G is a generator set for G if $G = \langle S \rangle$. Observe that, if S is a set of generators for G then G is abelian iff $\forall g_1, g_2, \in S : g_1 g_2 = g_2 g_1$.

Let H be a subgroup of a group G. For $g \in G$ the set $\{hg \mid h \in H\}$ denoted by Hg is a *right coset* of H in G. Similarly, the set $gH = \{gh \mid h \in H\}$ is a *left coset* of H in G.

A subgroup H of G is a *normal* subgroup of G if for all $g \in G$ it holds that $Hg = gH$. As a consequence it turns out that the set $G/H = \{Hg \mid g \in G\}$ is a group (the *quotient group* induced by H) under the binary operation $\cdot$ defined by $Hx \cdot Hy = Hxy$.

Let p be a prime. A p-group is a finite group whose order is a power of p. Let G be finite groups such that $|G| = p_1^{e_1} p_2^{e_2} \dots p_r^{e_r}$. A fundamental result in group theory, due to Sylow states that 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 $(x_1, y_1) \circ (x_2, y_2) = (x_1 * x_2, y_1 . y_2)$.

3. Prime-power group-definable languages

In this section, we prove the PP-lowness of group-definable languages, where the groups are a priori known to be p-groups for some prime p.

Using the simple fact that for any fixed prime p, the size of any p-subgroup of B_m can take only polynomially (in m) many values, we next show that p-group-definable language are in the class LWPP and hence low for PP.

Theorem 3.1. *For any fixed prime p, every p-group-definable language is in* LWPP *and hence low for* PP.

Proof. Let L be p-group-definable for a prime p over the group family $\mathscr{B} = \{B_m\}_{m\geqslant 1}$, via an NP machine M. We give an LWPP machine which accepts L.

```
DESCRIPTION OF MACHINE $M'$(x)
1   Compute $n = |B_{i(|x|)}|$
2   Find the greatest integer $\alpha$ such that $p^\alpha$ divides $n$
3   Produce a $gap = \prod_{j=1}^{\alpha} p^j - \prod_{j=1}^{\alpha}(p^j - acc_M(x))$
```

We claim that for $x \in L$ the *gap* produced by M' is $\prod_{j=1}^{\alpha} p^j$ and if $x \notin L$, the *gap* will be 0. Now, when $x \in L$, $Sol_R(x)$ will be a right coset of some p-subgroup, say G, of $B_{i(|x|)}$. By Lagrange's theorem, the order of G should be p^k for some $k \leqslant \alpha$. Hence $acc_M(x) = p^k$ and $\prod_{j=1}^{\alpha}(p^j - acc_M(x)) = 0$. So the total $gap = \prod_{j=1}^{\alpha} p^j$. If $x \notin L$, $acc_M(x) = 0$ and hence total $gap = 0$. Since, $\prod_{j=1}^{\alpha} p^j$ depends only on $|x|$ and computable in polynomial time for a fixed prime p, it follows that, $L \in$ LWPP. □

Next, we prove that prime-power group-definable languages are low for PP. Although, the proof is very similar to the proof of lowness of p-group-definable for PP, we are unable to get membership in LWPP. This is because the gap of the final machine (defined below) depends on the prime factorization of $|B_m|$ which is not known to be polynomial time computable.

The next theorem gives us a sufficient condition for a language to be low for PP. We use this theorem for proving lowness for PP of prime-power group-definable languages and abelian group-definable (see Section 4) languages. We omit the proof of this theorem which is exactly equal to the proof of that the class LWPP is low for PP given in [11].

Theorem 3.2. *Let L be a language such that there exists a* GapP *machine M and a* GapP *function $f : \mathbf{N} \to \mathbf{Z}$ with the property: for all $x \in L$, $gap_M(x) = f(1^{|x|})$ and for all $x \notin L$, $gap_M(x) = 0$. Then L is low for* PP.

Theorem 3.3. *Every prime-power group-definable language is low for* PP.

Proof. For the proof, we make use of the fact that the number of primes that divide $|B_m|$ is polynomially bounded in m, and for any prime p dividing $|B_m|$, the order of any p-group of B_m can take only polynomially many values.

Let L be prime-power group-definable over the group family $\mathscr{B} = \{B_m\}_{m\geqslant 1}$ via a relation R. Let M be the NP machine corresponding to R. We give a machine M' which satisfies the hypothesis of Theorem 3.2.

```
Description of Machine $M'(x)$
1   Compute $| B_{r(|x|)} | = n$
2   **Guess** $k : 1 \leqslant k \leqslant \log n$
3   **for** $i \leftarrow 1$ **to** k
4       **do Guess** integer $p_i > p_{i-1}$ and UP certificate $s_i$ of primality of $p_i$
5          **if** $p_i$ not prime
6             **then** produce a $gap=0$
7          **Guess** index $e_i$
8   **if** $n \neq p_1^{e_1} p_2^{e_2} \cdots p_k^{e_k}$
9     **then** produce a $gap=0$
10  Produce a $gap = \prod_{i=1}^{k}[\prod_{j=1}^{e_i} p_i^j - \prod_{j=1}^{e_i}(p_i^j - acc_M(x))]$
```

In *lines* 1–9, M' computes the prime factorization of $|B_{r(|x|)}| = n$ in a UP way using the algorithm in [10]. So, at the end of *line* 9, all the paths except one will produce a *gap*=0. Now, using similar arguments as in the proof of the above theorem, we can see that for $x \in L$ the *gap* produced by M' is $\prod_{i=1}^{k}\prod_{j=1}^{e_i} p_i^j$ and if $x \notin L$, the *gap* will be 0. It is easy to see that on input $1^{|x|}$, computing the prime factorization of $|B_{r(|x|)}|$ and hence $\prod_{i=1}^{k}\prod_{j=1}^{e_i} p_i^j$ is in GapP. So by Theorem 3.2, PP-lowness follows. $\square$

4. Abelian group-definable languages

In this section, we prove the main theorem that abelian group-definable languages are low for PP. Although we have defined the notion of abelian group-definable languages in the introduction, here we give a definition for the relativized version of this notion, since we have to use the relativized version of the main theorem for applying to specific problems.

Definition 4.1. A language $A \subseteq \Sigma^*$, is said to be abelian group-definable relative to an oracle $L \subseteq \Sigma^*$, if there is a group family $\mathscr{B}$ and a polynomial i such that, $A \in \mathrm{NP}(L)$ is defined by a relation $R \in \mathrm{P}(L)$, and for every $x \in A$ the set $Sol_R(x)$ is a right coset of some *abelian* subgroup of $B_{i(|x|)}$. If $L = \phi$, we say that A is abelian group-definable.

We first state some theorems from the theory of finite abelian groups which are essential in this section. The proofs of these theorems can be found in textbooks [8, 13]. The following representation theorem is a fundamental result.

Theorem 4.2 (Burnside [8]). *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. 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 $1 \leqslant i \leqslant r$. Furthermore, for $1 \leqslant i \leqslant r$, 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 $e_{i1} \geqslant e_{i2} \geqslant \cdots \geqslant e_{is_i}$ and $\sum_{j=1}^{s_i} e_{ij} = e_i$. This decomposition of G is unique.*

Let p be a prime and consider an abelian p-group G of order p^e. The above theorem implies that there is a *unique* sequence of natural numbers $e_1 \geqslant e_2 \geqslant \cdots \geqslant e_m$ such that $\sum_{1 \leqslant j \leqslant m} e_j = e$ and G can be expressed as a direct product of m cyclic groups of respective orders p^{e_j}, $1 \leqslant j \leqslant m$. The sequence $(e_1, e_2, \dots, e_m)$ is called the *type* of the p-group G. An element $g \in G$, $g \neq e$, is said to be *independent* of a set X of elements of G, if $\langle g \rangle \cap \langle X \rangle = \langle e \rangle$. A generator set X of G is an *independent generator set* for G if every $g \in X$ is independent of $X - \{g\}$. Let H 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. Let X_i be an independent generator set for the p_i-Sylow group. Then $\bigcup_{i=1}^{r} X_i$ is an *independent generator set* for H. As a consequence of Theorem 4.2, all finite abelian groups have independent generator sets.

For the proof of the main theorem, we need to know the number of independent generator sets of an abelian group. In the next subsection, we derive a formula for counting the number of independent generator sets of an abelian group.

4.1. Number of independent generator sets

In this subsection we derive a formula for counting the number of independent generators of an abelian p-group, for prime p, of given type.

Let p be a prime. Let G be an abelian p-group of order p^m. First we derive a formula for number of elements of G of order p^μ for any $1 \leqslant \mu \leqslant m$. Let $G_\mu = \{g \in G : \mathrm{o}(g) \leqslant p^\mu\}$. Then G_μ is a subgroup of G. The following lemma gives an easy way of computing the number of elements of order p^μ in G from the type $(m_1, m_2, \dots, m_s)$ of G.

Lemma 4.3 (Burnside [8]). *Let G be a* p-group *of type $(m_1, m_2, \dots, m_s)$. Let $G_\mu = \{g \in G \mid \mathrm{o}(g) \leqslant p^\mu\}$ and $|G_\mu|$ be p^ν. Then $\nu = k\mu + \sum_{i=k+1}^{s} m_i$ where k is such that $m_k \geqslant \mu \geqslant m_{k+1}$ and the number of elements of G of order p^μ is $|G_\mu| - |G_{\mu-1}|$.*

Now we state an easy result which allows us to adapt the above lemma for abelian factor p-groups.

Lemma 4.4 (Burnside [8]). *Let G be a* p-group *of type $(m_1, m_2, \dots, m_s)$. Let $g_1, g_2, \dots, g_s$ be an independent set of generators for G with $\mathrm{o}(g_i) = p^{m_i}$, $1 \leqslant i \leqslant s$. Let $G(i)$ be the subgroup of G generated by the set $\{g_1, g_2, \dots, g_i\}$. Then for every i, $1 \leqslant i \leqslant s$, the factor group $H^i = G/G(i)$ is of type $(m_{i+1}, m_{i+2}, \dots, m_s)$, and there are $|H^i_{m_{i+1}}| - |H^i_{m_{i+1}-1}|$ elements of order $p^{m_{i+1}}$ in H^i.*

Next we prove the crucial lemma which, along with the above two results, gives a formula for counting the number of independent generator sets of an abelian p-group from its type.

Lemma 4.5. *Let G be a finite abelian p-group of type $(m_1, m_2, \ldots, m_s)$. Let $g_1, g_2, \ldots, g_s$ be an independent set of generators for G with $\mathrm{o}(g_i) = p^{m_i}$, $1 \leqslant i \leqslant s$. Let $G(i)$ be the subgroup of G generated by the set $\{g_1, g_2, \ldots, g_i\}$. Let $H^i = G/G(i)$. Then the number of independent generator sets K of G is given by $K = \prod_{i=0}^{s}(|H^i_{m_{i+1}}| - |H^i_{m_{i+1}-1}|)p^{im_{i+1}}$.*

Proof. First, it follows from uniqueness part of fundamental theorem (Theorem 4.2) that if $\{g_1, g_2, \ldots, g_s\}$ is an independent generator set then $\mathrm{o}(g_i) = p^{m_i}$ for all $1 \leqslant j \leqslant r$. Suppose we already have an independent set $\{g_1, g_2, \ldots, g_i\}$ of elements such that $\mathrm{o}(g_j) = p^{m_j}$ for $1 \leqslant j \leqslant i$. So, if we are able to get the number of elements of order $p^{m_{i+1}}$ in G, which are independent to the set $\{g_1, g_2, \ldots, g_i\}$, then by induction we can get the number of independent generator sets for G. Hence we will concentrate on counting the number of elements of order $p^{m_{i+1}}$ in G, which are independent to the set $\{g_1, g_2, \ldots, g_i\}$,

From Lemma 4.4, it follows that there are $|H^i_{m_{i+1}}| - |H^i_{m_{i+1}-1}|$ elements (cosets of $G(i)$ in G) of order $p^{m_{i+1}}$ in H^i. So it follows that if we could count the number of elements of G of order $p^{m_{i+1}}$ in any right coset $G(i)g$ of order $p^{m_{i+1}}$ in G, we can calculate the number of elements of G of order $p^{m_{i+1}}$ independent to the set $\{g_1, g_2, \ldots, g_i\}$.

Consider a right coset $G(i)g$ of order $p^{m_{i+1}}$. It follows from the proof of Theorem 4.2 (described in [8]), that any right coset $G(i)g$ of order $p^{m_{i+1}}$ has an element g_{i+1} of order $p^{m_{i+1}}$ in G independent of $\{g_1, g_2, \ldots, g_i\}$. So, $G(i)g = G(i)g_{i+1}$ and we want the number of elements $h \in G(i)$ satisfying the equation: $(hg_{i+1})^{p^{m_{i+1}}} = e$. Notice that, since g_{i+1} is independent of $G(i)$, $(hg_{i+1})^j$ cannot be equal to identity for $j < p^{m_{i+1}}$. Since $\{g_1, g_2, \ldots, g_i\}$ are independent, for any $h \in G(i)$ there exists a unique set of indices $l_1, \ldots, l_i$; $l_j < p^{m_j}$ for $1 \leqslant j \leqslant i$ such that $h = g_1^{l_1} \ldots g_i^{l_i}$. So, since $(g_{i+1})^{p^{m_{i+1}}} = e$, the number of elements $h \in G(i)$ satisfying the condition $(hg_{i+1})^{p^{m_{i+1}}} = e$ is the same as the number of tuples $(l_1, \ldots, l_i)$; $l_j < p^{m_j}$ for $1 \leqslant j \leqslant i$ such that $g_j^{l_j p^{m_{i+1}}} = e$ for $1 \leqslant j \leqslant i$. Since the order of g_j is p^{m_j}, the above condition holds for all those l_j such that p^{m_j} divides $l_j p^{m_{i+1}}$. So for each j; $1 \leqslant j \leqslant i$, there are $p^{m_{i+1}}$ possibilities. This implies that the total number of tuples $(l_1, \ldots, l_i)$; satisfying the condition $g_j^{l_j p^{m_{i+1}}} = e$ for $1 \leqslant j \leqslant i$ is $p^{im_{i+1}}$.

So the number of elements in G of order $p^{m_{i+1}}$ independent of the set $\{g_1, g_2, \ldots, g_i\}$ are $(|H^i_{m_{i+1}}| - |H^i_{m_{i+1}-1}|)p^{im_{i+1}}$. Hence the lemma follows from induction. □

Finally, we extend the above lemma to get the number of independent generators for an arbitrary abelian group. The proof follows from the fact that each independent generator set for an abelian group can be written as a union of an independent generator set for its p-Sylow groups.

Lemma 4.6. *Let G be a finite abelian group of order $n = p_1^{e_1} p_2^{e_2} \dots p_k^{e_k}$. Let K_i be the number of independent generators of the p_i-Sylow subgroup of G. Then the number of independent generators K for G is given by $K = \prod_{i=1}^{k} K_i$*

4.2. Main theorem

In this subsection, we prove that abelian group-definable languages are low for the counting class PP. We introduce some notations and state a weaker version of a theorem from [15] which we use for the proof of the main theorem.

Let M be an oracle NP machine and let $A \in$ NP accepted by an NP machine N and let $f \in$ FP. We say that M^A make *1-guarded* queries if on any input x, M^A only asks queries y such that N on input y has either 0 or 1 accepting path.

Theorem 4.7 (Kobler et al. [15]). *Let M be an oracle* NP *machine. If $A \in$* NP *such that for all inputs M^A makes only 1-guarded queries to A then the function $h(x) = gap_{M^A}(x)$ is in* GapP.

Theorem 4.8. *Every abelian group-definable language is low for* PP.

Proof. Let $\mathscr{B} = \{B_l\}_{l>0}$ be a group family. Let L be an abelia n-group-definable language over $\mathscr{B}$, via the relation R, witnessed by the NP machine M, such that for $x \in L$, $Sol_R(x)$ is a right coset of an abelian subgroup, say G_x, of $B_{r(|x|)}$, where r is some polynomial. Let $|B_{r(|x|)}| = n$ whose prime factorization is $p_1^{e_1} p_2^{e_2} \dots p_k^{e_k}$. Let $G^{(i)}$ be the p_i-Sylow subgroup of G_x. We first give an intuitive sketch of the proof and then describe the details.

In order to prove PP-lowness of L, we design a machine N which has the property that for all $x \in L$, N produces a gap which depends only on $|x|$, which can also be computed in GapP; and for $x \notin L$, N produces a zero gap. Then PP-lowness of L follows from Theorem 3.2. For this, we crucially use the following fact. Since, for all $x \in L$, $|Sol_R(x)| = |G_x|$ divides $|B_{r(|x|)}|$ and number of distinct prime factors of $|B_{r(|x|)}|$ is small (polynomially in $|x|$), it follows that the number of prime factors in $|Sol_R(x)|$ for all x of same length collected together, is small.

Consider a nondeterministic machine N_A that starts its computation by simulating the NP machine that accepts L. If N_A could somehow compute the number $|Sol_R(x)| = |G_x|$ on each of its accepting paths, then it has to just branch into an appropriate number of accepting paths, so that the total number of accepting paths only depends on $|x|$.

Thus the crux of the proof is to design a suitable method for computing $|G_x|$, and build this method into N_A. Observe that by Theorem 4.2 $|G_x|$ can be computed if the type of each Sylow subgroup of G_x is known. Therefore, N_A guesses the types of each of the Sylow subgroups of G_x. In order to authenticate the guessed types, N_A further guesses an independent set of generators for each of the Sylow subgroups of G_x and computes the orders of each generator. Here, Lemma 4.6 which gives us a formula for the number of independent generator sets of an abelian p-group of given type comes in handy. Next, N_A is left with the task of verifying that each of the guessed generator sets are indeed independent. In order to do this verification N_A needs to use an oracle $\mathscr{I}$ (which is an NP language formally defined below). A delicate part of the design of N_A is that the queries it makes to $\mathscr{I}$ are 'guarded' in the following sense: for each query q to $\mathscr{I}$ made by N_A, it holds that the NP machine that accepts $\mathscr{I}$ has at most *one* accepting path. This 'guarded access' to $\mathscr{I}$ enables us to eventually do away with the oracle $\mathscr{I}$. This is accomplished by suitably applying Theorem 4.7.

Call a computation path of N_A *good* if along that path N_A guesses independent generator sets for each of the Sylow subgroups of G_x. One problem still left with N_A is that there may be computation paths along which one or more guessed independent generator set(s) generates only a *proper* subgroup of the corresponding Sylow subgroup of G_x. Call such paths as *bad* paths.

So, N_A as an oracle NP machine, actually accepts only the language defined as follows: $A = \{\langle x, w, p_i, m\rangle \mid w \in Sol_R(x)$; the p_i-Sylow subgroup $G^{(i)}$ of G_x has order divisible by $p_i^m\}$.

In order to get rid of the bad paths of N_A we design another machine N which simulates N_A in a suitable manner so that for each bad path of N_A the machine N produces a zero *gap*, and N produces a fixed nonzero gap for each good path of N_A.

We shall now turn to the formal details of the proof. First we shall define a language in $\mathscr{I} \in$ NP which can be used to test independence of elements in an abelian group. Define $\mathscr{I} = \{\langle 0^m, S, g\rangle | S \subseteq B_m; g \in B_m; \langle S\rangle$ is abelian and $g \in S\}$

Claim 4.8.1. There exists an NP-machine M which accepts $\mathscr{I}$. Moreover if S is an independent set of elements, then for those inputs $\langle 0^m, S, g\rangle \in \mathscr{I}$, M will have a unique accepting path.

Proof. Firstly, we note that if the prime factorization $p_1^{e_1} p_2^{e_2} \dots p_k^{e_k}$ of $|B_m|$ is given, then $\mathrm{o}(g)$ for $g \in B_m$, can be computed in time polynomial in m. To see this, observe that $\mathrm{o}(g)$ can be expressed as $p_1^{d_1} p_2^{d_2} \dots p_r^{d_r}$, each $d_i \leqslant e_i$. Clearly d_i is the smallest j such that $(g^{n/p_i^{e_i}})^{p_i^j} = e$. Since $1 \leqslant d_i \leqslant e_i$ 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 m.

The NP-machine M after verifying that $\langle S\rangle$ is abelian, computes $|B_m|$, guesses its prime factorization in a UP way [10], compute $\mathrm{o}(h)$ for all $h \in S$, guesses indices $1 \leqslant l_h \leqslant \mathrm{o}(h)$ and verifies that $g = \prod_{h \in S} h^{l_h}$. Now, if S is independent, then there will be a unique guess for l_h for all $h \in S$. So in this case M will have a unique accepting path. $\square$

Next, we define a language $A = \{\langle x, w, p_i, m\rangle \mid w \in Sol_R(x)$; the p_i-Sylow subgroup $G^{(i)}$ of G_x has order divisible by $p_i^m\}$.

For an integer n, define positive integers $L_i(n) = p_i^{e_i^2 + e_i} \prod_{1 \leqslant \alpha \leqslant e_i} (p_i^{\alpha} - 1)$, for $1 \leqslant i \leqslant k$, where $|B_{r(n)}| = p_1^{e_1} p_2^{e_2} \dots p_k^{e_k}$.

Claim 4.8.2. There is an oracle NP chine $N_A^{\mathscr{I}}$ that makes 1-guarded queries to $\mathscr{I}$ and accepts A. Furthermore, for inputs $\langle x, w, p_i, m\rangle$ in A, where $p_i^m = |G^{(i)}|$, $N_A^{\mathscr{I}}$ has exactly $L_i(|x|)$ accepting paths.

Proof. We will define an oracle NP machine $N_A^{\mathscr{I}}$ that makes 1-guarded queries to $\mathscr{I}$ and accepts A with the desired properties.

The idea is that $N_A^{\mathscr{I}}$ on input $\langle x, w, p_i, m\rangle$, guesses a type $\langle m_{1,i}, m_{2,i}, \dots, m_{k(i),i}\rangle$ and verifies that $m = \sum_{j=1}^{k(i)} m_{j,i}$. It then verifies that this is the type of a *subgroup* of $G^{(i)}$ by making suitable guarded queries to $\mathscr{I}$,

```
DESCRIPTION OF MACHINE $N_A^{\mathscr{I}}(x, w, p_i, m)$
1   if $w \notin Sol_R(x)$
2      then Reject
3   Guess the UP certificate for the primality of $p_i$ and check that $p_i$ is prime
4   if $p_i$ not prime
5      then Reject
6   Compute the following
7        (a) $n = |B_{r(|x|)}|$
8        (b) Index $e_i$ which is the highest $p_i$ power that divides $n$
9        (c) $L_i(r(|x|))$
10  (*All the above quantities can be computed in time polynomial in $|x|$ *)
11  Guess $k$: $1 \leqslant k \leqslant e_i$.
12  Guess the type $(m_1, m_2, \dots, m_k)$ where $1 \leqslant m_1, m_2, \dots, m_k \leqslant e_i$
13     if $(m \neq \sum_{j=1}^{k} m_j)$ or $(m_j > m_i$ for $j > i)$
14  then Reject
15  Using the type $(m_1, m_2, \dots, m_k)$ Compute $K_i$ as in Lemmas 4.6 and 4.3
16  Guess distinct strings $g_1, \dots, g_k \in \Sigma^{r(|x|)}$
17  if $(o(g_j) \neq p_i^{m_j})$or$(g_j w \notin Sol_R(x))$
18     then Reject
19  $j \leftarrow 2$
20  while $\langle 0^{r(|x|)}, \{g_1, \dots, g_{j-1}\}, g_j\rangle \notin \mathscr{I}$ and $j \leqslant k$ do $j \leftarrow j + 1$
21  if $j = k + 1$
22     then Branch into $L_i(|x|)/K_i$ paths and Accept
23     else Reject
```

It is easy to see that $N_A^{\mathscr{I}}$ accepts A. To see that the rest of the claim is correct, we first note that if the **while**-loop in *lines* 20 and 21 is exited with $j = k + 1$, then the **while**-loop condition guarantees that the guessed elements $g_1, g_2, \dots, g_k$ are independent. *Lines* 17 and 18 guarantees that each g_j has order p^{m_j} and are elements of $G^{(i)}$, $1 \leqslant j \leqslant k$. Also *lines* 13 and 14 ensures that $m = \sum_{j=1}^{k} m_j$. Therefore $\langle g_1, g_2, \dots, g_k\rangle$ is a subgroup of $G^{(i)}$ of order p_i^m.

Let us consider the case when $p_i^m = |G^{(i)}|$. If the **while**-loop in *line* 20 is exited with $j = k + 1$, then it holds that $\{g_1, g_2, \dots, g_k\}$ is a generator set for $G^{(i)}$ and, moreover, *lines* 21–23 give rise to $L_i(|x|)/K_i$ accepting paths. Furthermore, Theorem 4.2 guarantees that there is only a unique guess of the type $m_1 \geqslant m_2, \cdots \geqslant m_k$ that can lead to accepting paths. For that unique guess K_i is correctly computed in *line* 15. Also, from Lemma 4.6, we know that $G^{(i)}$ has K_i generator sets. Thus $N_A^{\mathscr{I}}$ has $K_i \cdot L_i(|x|)/K_i = L_i(|x|)$ accepting paths on input $\langle x, w, p_i, m\rangle$ when p_i^m is $|G^{(i)}|$.

To see that the machine $N_A^{\mathscr{I}}$ is making 1-guarded queries to $\mathscr{I}$, notice that $N_A^{\mathscr{I}}$ is querying $\mathscr{I}$ only in *line*-20. Suppose at the jth iteration of the **while**-loop, for a query $\langle 0^{r(|x|)}, \{g_1, \dots, g_{j-1}\}, g_j\rangle$ to $\mathscr{I}$, the machine M described in claim 4.8.1 has more than one accepting path. This means that the set $\{g_1, \dots, g_{j-1}\}$ is not independent. That is there exists g_i; $i < j$ such that $g_i \in \langle g_1, \dots, g_{i-1}\rangle$. This is a contradiction to the fact that the machine has not exited the **while**-loop at the ith iteration. Hence the queries made by $N_A^{\mathscr{I}}$ to I are 1-guarded. □

Now, using the Theorem 4.7, we can replace the oracle machine $N_A^{\mathscr{I}}$ with a GapP machine N_A', which produces the same gap as the number of accepting paths of $N_A^{\mathscr{I}}$. The following claim is a direct application of Theorem 4.2 to Claim 4.2.

Claim 4.8.3. There is a GapP machine N_A' defining a GapP function h such that the language A is $\{\langle x, w, p_i, m\rangle \mid h(\langle x, w, p_i, m\rangle) \neq 0\}$. Furthermore, for inputs $\langle x, w, p_i, m\rangle$ in A, if $p_i^m = |G^{(i)}|$ then $h(\langle x, w, p_i, m\rangle) = L_i(|x|)$.

In the remaining part of the proof, we design a GapP machine N which on input $x \in \Sigma^*$, uses N_A' and has the following accepting behavior. For all $x \in L$, N will have a $gap = n\prod_{i=1}^{k} L_i^{e_i}(|x|)$ which depends only on $|x|$ and for all $x \notin L$, N will produce a 0 *gap*. Once existence of such a machine for L is being proved, the PP-lowness of L follows.

Claim 4.8.4. There exists a GapP machine N which on input $x \in \Sigma^*$ has the following accepting behavior. For all $x \in L$, N will produce a $gap = n\prod_{i=1}^{k} L_i^{e_i}(|x|)$ and for all $x \notin L$, N will produce a $gap = 0$.

We shall first give the description of machine N. Then we shall prove that N has the above mentioned accepting behavior.

Description of N(x)

```
1   Compute $| B_{r(|x|)} | = n$
2   **Guess** $k : 1 \leqslant k \leqslant \log n$
3   **for** $i \leftarrow 1$ **to** k
4       **do Guess** integer $p_i > p_{i-1}$ and UP certificate $s_i$ of primality of $p_i$
5          **if** $p_i$ not prime
6             **then** produce a $gap$=0
7          **Guess** index $e_i$
8   **if** $n \neq p_1^{e_1} p_2^{e_2} \dots p_k^{e_k}$
9      **then** produce a $gap$=0
10  Guess $w \in \Sigma^{r(|x|)}$
11  **if** $w \notin Sol_R(x)$
12     **then** produce a $gap$=0
```

```
13  for 1 ← 1 to $k$
14    do Guess $l_i$ : $1 \leqslant l_i \leqslant e_i$
15  Compute $\beta = \prod_{i=1}^{k} p_i^{l_i}$
16  Branch into $n/\beta$ paths and produce a gap of
17  $\prod_{i=1}^{k}[h(\langle x, w, p_i, l_i\rangle) \cdot L_i^{l_i-1}(x) \cdot \prod_{m=l_i+1}^{e_i}(L_i(x) - h(\langle x, w, p_i, m\rangle))]$
```

From the definition of the group family, it follows that the computation at *line* 1 is polynomial time. Since, checking whether an integer is prime or not is in the class UP [10], *lines* 2–10 of the machine ensures that after the computation of *line* 10, N will have the unique prime factorization of n on exactly one of its path and all other paths will produce a *gap*=0. Now, we shall see the accepting behavior of N on this path.

Recall that h is the GapP function computed by N'_A defined in Claim 4.2. Consider the gap $\prod_{i=1}^{k}[h(\langle x, w, p_i, l_i\rangle) \cdot L_i^{l_i-1}(x) \cdot \prod_{m=l_i+1}^{e_i}(L_i(x) - h(\langle x, w, p_i, m\rangle))]$ that machine N produces in *lines* 9 and 10. Let us denote the gap by X, for brevity. Consider the case when, for each $1 \leqslant i \leqslant k$, the value l_i guessed in *lines* 6 and 7 is m_i, where $p_i^{m_i}$ is the size of the p_i-Sylow subgroup $G^{(i)}$ of G_x. Then it holds that $p_i^{l_i} = |G^{(i)}|$ for each i, $1 \leqslant i \leqslant k$. In this case $h(\langle x, w, p_i, l_i\rangle) = h(\langle x, w, p_i, m_i\rangle) = L_i(x)$ for all $1 \leqslant i \leqslant k$ (Claim 4.8.3) and $h(\langle x, w, p_i, m\rangle) = 0$ for all $m > l_i$. So the gap corresponding to this particular set of guesses for l_i, $1 \leqslant i \leqslant k$, is $\prod_{i=1}^{k} L_i(x)^{e_i}$. On the other hand, consider the case when one of the guessed values l_i is different from the corresponding correct value m_i. For that index i, if $l_i > m_i$ then $h(\langle x, w, p_i, l_i\rangle) = 0$, and if $l_i < m_i$ then $\prod_{m=l_i+1}^{e_i}(L_i(x) - h(\langle x, w, p_i, m\rangle)) = 0$ since $L_i(x) - h(\langle x, w, p_i, m_i\rangle) = 0$, and $m_i \geqslant l_i + 1$. Thus, for an incorrect guess $l_1, l_2, \ldots, l_k$, the corresponding gap X will take the value 0. Therefore, the gap X evaluates to 0 (for each of the n/β paths into which it branches at *line* 9) if the guess $l_1, l_2, \ldots, l_k$ is incorrect. On the other hand, the gap X evaluates to $\prod_{i=1}^{k} L_i^{e_i}$ for the unique correct guess $m_1, m_2, \ldots, m_k$. Thus, the total gap produced under the correct guess of $m_1, m_2, \ldots, m_k$ is $|G_x| \cdot (n/|G_x|) \cdot \prod_{i=1}^{k} L_i^{e_i}$ which is $n \prod_{i=1}^{k} L_i^{e_i}$. Hence the claim. □

Now to complete the proof of the theorem, we note that it is easy to design a *gap* machine which on input $1^{|x|}$ produces a gap $n \prod_{i=1}^{k} L_i^{e_i}$, where $n = |B_{r(|x|)}|$. So by Theorem 3.2 and Claim 4.8.4, it follows that L is low for PP. □

It is routine to verify that the above proof relativizes. We state a relativized version of the theorem.

Theorem 4.9. *Let $L \subseteq \Sigma^*$. Then every language that is abelian group-definable relative to L is low for* PP(L).

4.3. Applications of the main theorem

Now we shall see the consequence of the above theorem on the PP-lowness for several abelian black-box group problems. These problems are well-studied in computational group theory [3, 5, 9, 15]. Here we would like to remark that in another paper [1], we actually show that all the problems we are considering here are low for PP, using proof methods specific to these problems. What is interesting here is that Theorem 4.8 places various abelian group problems in a uniform framework, and their PP-lowness follows directly.

The problems that we are interested in are Membership Testing, Group Factorization, Coset Intersection and Double Coset Factorization. We formally define these problems for general black-box groups, over an arbitrary group family $\mathscr{B}$. We are interested in the case when these groups are abelian. Let c be a polynomial.

Bounded-Membership $\overset{\text{def}}{=} \{\langle 0^m, S, g\rangle | S \subseteq B_m; g \in B_m; |S| \leqslant c(m);\ g \in \langle S\rangle\}$,
Membership $\overset{\text{def}}{=} \{\langle 0^m, S, g\rangle | S \subseteq B_m; g \in B_m; g \in \langle S\rangle\}$,
Group-Factorization $\overset{\text{def}}{=} \{(0^m, S_1, S_2, g) \mid S_1, S_2 \subseteq B_m \text{ and } g \in \langle S_1\rangle\langle S_2\rangle\}$,
Coset-Intersection $\overset{\text{def}}{=} \{(0^m, S_1, S_2, g) \mid S_1, S_2 \subseteq B_m \text{ and } \langle S_1\rangle g \cap \langle S_2\rangle \neq \emptyset\}$.
Double-Coset-Membership $\overset{\text{def}}{=} \{(0^m, S_1, S_2, g, x) \mid S_1, S_2 \subseteq B_m \text{ and } g \in \langle S_1\rangle x \langle S_2\rangle\}$.

We first show that Bounded-Membership for abelian groups over an arbitrary group family $\mathscr{B}$ is abelian group-definable over a suitably chosen group family $\mathscr{X}$ and hence low for PP. Then we show that Membership is polynomial time Turing reducible to Bounded-Membership thereby implying that Membership is also low for PP.

For any integer r, let $\mathbf{Z}_r^+$ denote the additive group of integers modulo r. Let $\mathscr{Z}_{\langle r,s\rangle}$ denote the product group $\mathbf{Z}_r^+ \times \mathbf{Z}_r^+ \times \cdots \times \mathbf{Z}_r^+$, the product taken s times. Define the group family $\mathscr{X} = \{\mathscr{Z}_{\langle r,s\rangle}\}_{r,s\geqslant 1}$.

Proposition 4.10. *Let $\mathscr{B}$ be a group family. Then* Bounded-Membership *for abelian groups is abelian group-definable over $\mathscr{X}$.*

Proof. Let $\langle 0^m, \{g_1, g_2, \ldots, g_k\}, g\rangle$ be an instance of Bounded-Membership such that $k \leqslant c(m)$. By definition of group families, we know that $|B_m| = n$ is computable in time polynomial in m. So Bounded-Membership is in NP via the polynomial-time relation $R = \{\langle x, e_1, e_2, \ldots, e_{c(m)}\rangle | e_i \in \mathbf{Z}_n^+ \text{ for } 1 \leqslant i \leqslant c(m) \text{ and } g_1^{e_1} g_2^{e_2} \ldots g_k^{e_k} = x\}$. Now, $Sol_R(x) = \{\langle e_1, e_2, \ldots, e_{c(m)}\rangle \mid g_1^{e_1} g_2^{e_2} \ldots g_k^{e_k} = x \text{ and } e_i \in \mathbf{Z}_n^+ \text{ for } 1 \leqslant i \leqslant c(m)\}$. It is easy to see that this set is a right coset of the group $G = \{\langle e_1, e_2, \ldots, e_{c(m)}\rangle \mid g_1^{e_1} g_2^{e_2} \ldots g_k^{e_k} = id\}$ which is a subgroup of $\mathscr{Z}_{\langle n, c(m)\rangle}$. □

It immediately follows from Theorem 4.8 and Proposition 4.10 that the restricted abelian group membership problem is low for PP. We state this as next corollary.

Corollary 4.11. *Let $\mathscr{B}$ be a group family. Then* Bounded-Membership *for abelian groups is low for PP.*

Proposition 4.12. *Let $\mathscr{B}$ be a group family. Then* Membership *is polynomial-time Turing reducible to* Bounded-Membership.

Proof. Let $(0^m, S, g)$ be an instance of the Membership. From the definition of group families there is a polynomial p such that $|B_m| \leqslant 2^{p(|x|)}$. Using the following polynomial-time oracle machine with the Bounded-Membership as oracle, we can construct a generator set for A whose size is at most $p(|x|)$, and thereby reduce Membership to Bounded-Membership.

```
Reduce $(0^m, \{g_1, \ldots, g_k\}, g)$
1   $X \leftarrow \{g_1\}$
2   **for** $i \leftarrow 2$ **to** $k$
3       **do if** $(0^m, X, g_i) \notin$ Bounded-Membership
4           **then** $X \leftarrow X \cup \{g_i\}$
5   **if** $(0^m, X, g) \in$ Bounded-Membership
6       **then Accept**
```

It is easy to see that the set X computed by the **for**-loop in the above algorithm generates $\langle\{g_1, \ldots, g_k\}\rangle$. Furthermore, in the above algorithm, each time we include an element in X, by Lagrange's theorem, the size of the corresponding subgroup generated by it at least doubles in cardinality. Since $|B_m| \leqslant 2^{p(|x|)}$, it must hold that $|X| \leqslant p(|x|)$. This completes the proof. □

The next corollary immediately follows.

Corollary 4.13. *Let $\mathscr{B}$ be a group family. Then* Membership *for abelian groups is low for PP.*

Theorem 4.14. *Let $\mathscr{B}$ be a group family. Then the problems* Group-Factorization, Coset-Intersection *and* Double-Coset-Intersection *with respect to $\mathscr{B}$, for abelian groups are low for PP.*

Proof. It suffices to prove the result for Group-Factorization since Coset-Intersection and Double-Coset-Intersection are many-one reducible to Group-Factorization.

Consider an instance $(0^m, S_1, S_2, g)$ of Group-Factorization for abelian groups with respect to the group family $\mathscr{B}$, where $\langle S_1\rangle$ and $\langle S_2\rangle$ are abelian subgroups of B_m.

Consider the relation $R = \{(g_1, S_1, S_2, g) \mid g_1 \in \langle S_1\rangle$ and $g_1^{-1}g \in \langle S_2\rangle\}$. It is easy to see that $R \in$ P(Memb), for, the polynomial time machine has to query Membership for $(0^n, S_1, g_1)$ and $(0^n, S_2, g_1^{-1}g)$. Further, observe that corresponding to the relation R, the solution set of an instance $x = (0^m, S_1, S_2, g)$ is $Sol_R(x) = \{g_1 \mid g_1 \in \langle S_1\rangle$ and $g_1^{-1}g \in \langle S_2\rangle\}$. It is easy to see that $Sol_R(x)$ is a right coset of $\langle S_1\rangle \cap \langle S_2\rangle$ which is also an abelian group since $\langle S_1\rangle$ and $\langle S_2\rangle$ are abelian. This implies that Group-Fact is abelian group-definable relative to Membership. So from Theorem 4.9, it follows that Group-Factorization is low for PP(Membership). But from Corollary 4.13 we know that PP(Membership) = PP. Hence the theorem. □

5. Counting solutions

In this section we prove upper bounds on the complexity of the exact counting problem corresponding to group-definable languages. We show that, for group-definable languages, counting the number of solutions can be done in $\mathrm{FP}^{\mathrm{AM}}$ and particularly, for group-definable languages over the group family is SYM, counting the number of solutions can be done in $\mathrm{FP}^{\mathrm{NP}}$ with parallel queries.

As an intermediate result, we prove upper bounds on the following generalization of a problem posed by Hoffman [14]: Given a polynomial-time membership test for a subgroup H of a group G (presented by a set of generators), to compute a set of generators for H. An interesting aspect of this problem is that it is a kind of converse of the usual problem of membership testing in a group presented by a generator set.

Before we prove these results, we state some results from [3].

Theorem 5.1 (Babai [3]). *Let $\mathscr{B} = \{B_m\}_{m\geqslant 1}$ be a group family. Then the following sets are in* AM.
1. *Non-Membership* $\overset{\text{def}}{=} \{(0^m, S, g) \mid S \subseteq B_m, g \in B_m$ *and* $g \notin \langle S\rangle\}$.
2. *Order Verification* $\overset{\text{def}}{=} \{(0^m, S, n) \mid S \subseteq B_m$ *and* $|\langle S\rangle| = n\}$.

The following theorem gives an upper bound on computing a generator set for a group for which membership testing can be done in polynomial time.

Theorem 5.2. *Let $\mathscr{B} = \{B_m\}_{\geqslant m}$ be a group family. Let $H < B_m$. Then if a polynomial-time membership test is given for H, there is an* $\mathrm{FP}^{\mathrm{AM}}$ *algorithm for computing a generator set for H. If $\mathscr{B} =$* SYM, *then there is an* $\mathrm{FP}^{\mathrm{NP}}$ *algorithm for computing a generator set for H.*

Proof. Let $\mathscr{B} = \{B_m\}_{\geqslant m}$ be a group family. Let A be the be defined as $A = \{\langle 0^m, S, g'\rangle \mid \exists y \in \Sigma^{p(m)-|g'|};\ S \subseteq H;\ g'y \in H$ and $g'y \notin \langle S\rangle\}$ where p is a polynomial. First, we have the following easy claim.

Claim 5.2.1. $A \in$ AM. In particular, if the group family $\mathscr{B} =$ SYM, $A \in NP$.

Proof. First note that $A \in \exists$AM. This follows from part 1 of Theorem 5.1 and the fact that membership in H can be done in polynomial-time. Since $\exists$AM = AM, the claim follows for general black-box groups. In the case when the group family is SYM, the claim follows from the fact that, membership testing for permutation groups can be done in polynomial-time [12] (and hence non-membership testing also). $\square$

We next give a description of a deterministic algorithm which uses A to compute a generator set of H. Since, in general $A \in$ AM and in the case when the group family is SYM $A \in$ NP, the theorem follows.

```
Gen-Compute($0^m$)
1   $S \leftarrow \{e\}$
2   **while** $\langle 0^m, S, \varepsilon\rangle \in A$
```

```
3      do $g \leftarrow \varepsilon$
4         while $\langle 0^m, S, g\rangle \in A$
5            if $\langle 0^m, S, g1\rangle \in A$
6               $g \leftarrow g1$
7               else $g \leftarrow g0$
8         end-while (*Computes $g \in H - \langle S\rangle$ by prefix search*)
9         $S \leftarrow S \cup \{g\}$
10     end-while (* Computes a generator set for $G_x$*)
11     returns S
```

We shall see that the above algorithm computes a generator set for H in polynomial time. Note that the algorithm returns a set S when the **while**-loop condition of *line*-2 is violated. From the definition of the language A, this happens when $\langle S\rangle = H$. This shows that the algorithm computes a generator set for H. Note that after each iteration of **while**-loop of *line*-2, an element $g \in G$ but not in $\langle S\rangle$ is constructed and added to the set S before the next iteration. From Lagrange's theorem, it follows that $|\langle S \cup g\rangle| \geqslant 2|\langle S\rangle|$ if $g \notin S$. So it follows that the **while**-loop of *line*-2 will be iterated at most $p(m)$ times where $p(m)$ is the length of strings encoding the elements of B_m. □

As an interesting corollary to the above theorem, we prove upper bound on the complexity of the following problem: Given two black-box groups A and B presented by a generator set; compute a generator set for the group $A \cap B$.

Corollary 5.3. *Let $\mathscr{B} = \{B_m\}_{\geqslant m}$ be a group family. Let $A, B \subseteq B_m$, then a generator set for $\langle A\rangle \cap \langle B\rangle$ can be computed in* $\mathrm{FP}^{\mathrm{AM}}$.

Proof (*Sketch*). Define a language L as follows: $A = \{\langle 0^m, S, A, B, g'\rangle \mid \exists y \in \Sigma^{p(m)-|g'|};\ S \subseteq \langle A\rangle \cap \langle B\rangle;\ g'y \in \langle A\rangle \cap \langle B\rangle$ and $g'y \notin \langle S\rangle\}$. (Notice that the only difference between the language A in the above theorem and L is that the H in the definition of A is replaced by $\langle A\rangle \cap \langle B\rangle$ in L.)

In [5] it is shown that the problem of membership testing in a black-box group presented by a generator set is in NP. So from part 1 of Theorem 5.1, it follows that Membership Testing is in NP∩co-AM. From [17] we know that NP∩co-AM is low for AM. So it follows that the language $L \in AM$. So in the algorithm GEN-COMOPUTE, if we replace the language A by L, we will get an algorithm for computing a generator set for $\langle A\rangle \cap \langle B\rangle$. □

We use the algorithm GEN-COMPUTE to compute the #P function corresponding to a group-definable language. We do it in two parts. In the first part we show that for any input in the language, a generator set for the group whose right coset is the solution set, can be computed in $\mathrm{FP}^{\mathrm{AM}}$. In the next part, we use this set of generators to compute the corresponding #P function.

Lemma 5.4. *Let L be group-definable over a group family $\mathcal{B}$, via a polynomial-time computable relation R. For an input $x \in L$, let $G_x < B_m$ denote the group whose right coset is $Sol_R(x)$. Then a set of generators for G_x can be computed in* $\mathrm{FP}^{\mathrm{AM}}$.

Proof (*Sketch*). The algorithm is very similar to the algorithm GEN-COMPUTE. On an input x, using prefix search, first compute a solution $y \in Sol_R(x)$. This can be done in $\mathrm{FP}^L \subseteq \mathrm{FP}^{\mathrm{NP}}$ (since $L \in \mathrm{NP}$). Now, membership in G_x can be done in polynomial time, since checking whether an element g is in G_x is equivalent to checking whether $yg \in Sol_R(x)$, which can be done in polynomial time. From Theorem 5.2 it follows that a generator set can be computed in $\mathrm{FP}^{\mathrm{AM}}$.

Theorem 5.5. *Let L be a group-definable language over the group family, via the polynomial-time computable relation R. Then, $\forall x, |Sol_R(x)|$ can be computed in* $\mathrm{FP}^{\mathrm{AM}}$.

Proof. Given $x \in \Sigma^*$ as input, we describe an $\mathrm{FP}^{\mathrm{AM}}$ algorithm for computing the #P function $|Sol_R(x)|$. Let p be the polynomial such that $Sol_R(x) \subseteq \Sigma^{p(|x|)}$. For input x, let $G_x < B_m$ denote the group whose right coset is $Sol_R(x)$. Note that $|G_x| = |Sol_R(x)|$. Let $B' = \{\langle n, S\rangle \mid \text{n is the order of the group } \langle S\rangle\}$. The corresponding prefix language B is defined as: $B = \{\langle m, S\rangle \mid \exists n \in \Sigma^{\leqslant(p(|x|)-|m|)} mn \text{ is the order of the group } \langle S\rangle\}$. From part 2 of Theorem 5.1 and the fact that $\exists\mathrm{AM} = \mathrm{AM}$, it follows that $B \in AM$.

Now we give an algorithm Exact-Count, which on input x, first simulates the $\mathrm{FP}^{\mathrm{AM}}$ of Theorem 5.4 to first compute a generator set for G_x and then using the language B as oracle, computes $|G_x|$ in polynomial time.

```
Exact-Count($x$)
1 Simulating the algorithm in Theorem 5.4 compute a generator set for $G_x$ in $\mathrm{FP}^{\mathrm{AM}}$
2 $m \leftarrow \varepsilon$
3 while $\langle m, S\rangle \in B$
4       do if $\langle m1, S\rangle \in B$
5          then $m \leftarrow m1$
6          else  $m \leftarrow m0$
7    end-while (* Computes $|G_x|$ by prefix search*)
8 return m
```

From the description of the algorithm, it is clear that it computes the order of G_x in polynomial time using a language in AM. □

The above theorem along with Toda's theorem [18]that $\mathrm{PH} \subseteq \mathrm{P}^{\#\mathrm{P}}$ yield the following corollary which states that counting the number of solutions of instances of group-definable languages cannot be #P-complete unless the polynomial-time hierarchy collapses.

Corollary 5.6. *Let L be a group-definable language over the group family $\mathscr{B}$, w.r.t. a polynomial-time computable relation R. Then, if the function $f(x) = |Sol_R(x)|$ is* #P-*complete then* $\mathrm{PH} \subseteq \mathrm{P}^{\mathrm{AM}}$.

Next, we observe that in the special case where the group family is SYM, Theorem5.5 can be improved to get an $\mathrm{FP}_{||}^{\mathrm{NP}}$ algorithm (polynomial-time machine which makes parallel queries to an NP set) to compute the #P function. We omit the proof which is essentially the same as that of Mathon's result [16] that the number of isomorphisms between two graphs can be computed in polynomial time with parallel queries to GI.

Theorem 5.7. *Let L be a group-definable language over* SYM *with respect to a polynomial-time relation R. Then $\forall x$, $|\ Sol_R(x)\ |$ can be computed in* $\mathrm{FP}_{||}^{\mathrm{NP}}$.

References

[1] V. Arvind, N.V. Vinodchandran, solvable black-box group problems are low for PP, Symp. on Theoretical Aspects of Computer Science, Lecture Notes in Computer Science, vol. 1046, 1996, pp. 99–110.

[2] L. Babai, Trading group theory for randomness, 17th ACM symp. on Theory of Computing 1985, pp. 421–429.

[3] L. Babai, Bounded round interactive proofs in finite groups, SIAM J. Discrete Math. 5 (1992) 88–111.

[4] L. Babai, S. Moran, Arthur–Merlin games: a randomized proof system, and a hierarchy of complexity classes, J. Comput. System Sci. 36 (1988) 254–276.

[5] L. Babai, M. Szemerédi, On the complexity of matrix group problems I, Proc. 25th IEEE FOCS, 1984, pp. 229–240.

[6] J.L. Balcázar, J. Díaz, J. Gabarró, Structural Complexity – I & II, Springer, Berlin, 1988.

[7] R. Boppana, J. Hastad, S. Zachos, Does coNP have short interactive proofs? Inform. Process. Lett. 25 (1987) 127–132.

[8] W. Burnside, Theory of Groups of Finite Order, Dover, New York, 1955.

[9] G. Cooperman, L. Finkelstein, Combinatorial tools for computational group theory, DIMACS Series in Discrete Mathematics and Theoretical Computer Science, vol. 11, 1993.

[10] M. Fellows, N. Koblitz, Self-witnessing polynomial time complexity and prime factorization, Proc. 6th Structure in Complexity Theory Conf., 1992, pp. 107–110.

[11] S. Fenner, L. Fortnow, S. Kurtz, Gap-definable counting classes, Proc. 6th Structure in Complexity Theory Conf., 1991, pp. 30–42.

[12] M. Furst, J. E. Hopcroft, E. Luks, Polynomial time algorithms for permutation groups, Proc. 21st IEEE Symp. of Foundations of Computer Science, 1980, pp. 36–45.

[13] M. Hall, The Theory of Groups, Macmillan, New York, 1959.

[14] C. Hoffmann, Group-theoretic algorithms and graph isomorphism, Lecture Notes in Computer Science, vol. 136, Springer, Berlin, 1982.

[15] J. Köbler, U. Schöning, J. Torán, Graph isomorphism is low for PP, J. of Comput. Complexity 2 (1992) 301–310.

[16] R. Mathon, A note on graph isomorphism counting problem, Inform. Process. Lett. 8 (1979) 131–132.

[17] U. Schöning, Graph isomorphism is in the low hierarchy, J. Comput. System Sci. 37 (1988) 312–323.

[18] S. Toda, PP is as hard as polynomial-time hierarchy, SIAM J. Comput. 20(5) (1991) 865–877.