DATA STRUCTURES AND ALGORITHMS FOR COMPUTING IN NILPOTENT AND SOLVABLE PERMUTATION GROUPS

by

FERENC RÁKÓCZI

A DISSERTATION

Presented to the Department of Computer
and Information Science
and the Graduate School of the University of Oregon
in partial fulfillment of the requirements
for the degree of
Doctor of Philosophy

December 1997

“ Data Structures and Algorithms for Computing in Nilpotent and Solvable Permutation Groups,” a dissertation prepared by Ferenc Rákóczi in partial fulfillment of the requirements for the Doctor of Philosophy degree in the Department of Computer and Information Science. This dissertation has been approved and accepted by:

Eugene M. Luks

Chair of the Examining Committee

December 17, 1997

Date

Commitee in charge:	Dr. Eugene M. Luks, Chair
	Dr. Andrzej Proskurowski
	Dr. Christopher Wilson
	Dr. Charles R. B. Wright

Accepted by:

[illegible]

Vice Provost and Dean of the Graduate School

An Abstract of the Dissertation of

Ferenc Rákóczi for the degree of Doctor of Philosophy

in the Department of Computer and Information Science

to be taken December 1997

Title: DATA STRUCTURES AND ALGORITHMS FOR COMPUTING IN NILPOTENT AND SOLVABLE PERMUTATION GROUPS

Approved: Eugene M Luks

Dr. Eugene M. Luks

Computer algebra systems, such as GAP and Magma, are widely used for studying groups. Practical algorithms underlying these systems have been under development for over 30 years. More recently, there have been deep theoretical investigations into the asymptotic complexity of group-theoretic problems. The results have included demonstrations of polynomial-time solvability of problems whose traditional implementation, though usually efficient, would require exponential time in the worst case. This dissertation focuses on deterministic algorithms that meet both practical and theoretical standards of efficiency.

Most permutation-group computation employs a point-stabilizer series, a data structure first suggested by Sims in the 1960s. Variations by Knuth and Jerrum of Sims's basic algorithm had been shown to run in worst-case time $O(sn^2 + n^5)$, where n is the size of the permutation domain and s is the number of generators for the group. Certain important questions about the permutation group G , however, can be answered substantially faster than the time needed for just setting up Sims's data structure. We present such fast algorithms. Testing nilpotence of a group is shown to have deterministic time complexity $O(sn \log n \log^* n)$. Solvability is shown to be testable in time $O(sn^2)$.

Data structures are developed for computations in the families of nilpotent and solvable permutation groups. While reflecting the normal series that characterize the respective group family, these data structures are also naturally constructed and viewed within the permutation domain. Furthermore, they can be computed faster than the point-stabilizer series. The effectiveness of the data structures is demonstrated in their facilitation of algorithms that are based on the use of normal series.

For subgroups G and H of a nilpotent group, we consider computation of the following subgroups: the normalizer of H in G; the intersection of H and G; the centralizer of H in G.

The use of the data structure for solvable groups is illustrated in the implementation of a method for finding Sylow subgroups. It makes essential use of the vector space representation of the factors in the normal series.

All algorithms have been implemented in GAP and proved to perform well in practice, especially the recognition algorithms.

CURRICULUM VITA

NAME OF THE AUTHOR: Ferenc Rákóczi

PLACE OF BIRTH: Budapest, Hungary

DATE OF BIRTH: August 18, 1958

GRADUATE AND UNDERGRADUATE SCHOOLS ATTENDED:

University of Oregon,
Eõtvõs University, Budapest, Hungary

DEGREES AWARDED:

Doctor of Philosophy in Computer and Information Science, 1997, University of Oregon
Master of Science in Computer Science, 1992, University of Oregon
Diploma in Mathematics, 1982, Eõtvõs University

AREAS OF SPECIAL INTEREST:

Computational Algebra
Data Structures
Algorithms

PROFESSIONAL EXPERIENCE:

Teaching and Reasearch Assistant, Department of Computer and Information Science, University of Oregon, Eugene, 1990-1997

Research Assistant, Computer and Automation Institute of the Hungarian Academy of Sciences, 1982-1990

AWARDS AND HONORS:

Member of 6th place team, ACM International Collegiate Programming Contest, 1991

PUBLICATIONS:

E. M. Luks, F. Rákóczi and C. R. B. Wright, *Some Algorithms for Nilpotent Permutation Groups*, J. Symbolic Computation 23 (1997), 335–354.

E. M. Luks, F. Rákóczi and C. R. B. Wright, *Computing Normalizers in Permutation p-groups*, Proceedings of the International Symposium on Symbolic and Algebraic Computation (ISSAC), Oxford, July 1994, pp. 139–146; University of Oregon Technical Report CIS-TR-93-12.

G.W. Meyer, L.S. Peting and F. Rákóczi *A Color Gamut Visualization Tool*, IS&T/SID Color Imaging Conference: Transforms & Transportablity of Color (1993), pp. 197-201.

F. Rákóczi, *Fast Recognition of the Nilpotency of Permutation Groups*, Proceedings of the International Symposium on Symbolic and Algebraic Computation (ISSAC), Montreal, July 1995, pp. 265-269.

E. Tick, B.C. Massey, F. Rákóczi, and P. Tulayathun, *Concurrent Logic Programs a la Mode*, In Workshop on Practical Implementations and Systems Experience in Logic Programming. Budapest, June 1993; University of Oregon Technical Report CIS-TR-93-12.

ACKNOWLEDGEMENTS

First of all, I thank my advisor, Eugene Luks for his invaluable help and guidance that he provided during my years at the University of Oregon, for the uncounted hours he spent explaining concepts, algorithms and listening to and correcting my ideas about groups, algorithms, writing style and others. I am also indebted to Charley Wright, who was always easy to reach with questions, and whose explanations shed light for me to many topics in group theory.

I am also grateful to the other two members of my dissertation committee, Andrzej Proskurowski and Chris Wilson, for their suggestions that helped me to improve the readability of this dissertation.

During my years at the Department, a lot of people helped me to feel better, the office staff, especially Betty Lockwood and Jan Saunders, fellow students, who became good friends, as Takunari Miyazaki, Sastry, Jan Arne Telle, Amitabha Roy, Kevin Glass, Chris Harrop, Inkyu Kim. I am also very grateful to all of my office mates who put up with my unorderlyness, Sara Chieco, Sue Blair, Jens Mache, Karthika Devi and Peter Miller.

Last, but not at all least, I thank for the support of my family, my parents, brother and sister, my son and his mother.

DEDICATION

I dedicate this dissertation to my parents who never stopped supporting me.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER I

INTRODUCTION

History

The roots of computational group theory go back to the last century, when mathematicians, including Hölder and Mathieu, were using hand calculations for determining certain types of groups (the groups of order pqr and the first sporadic groups, respectively). It was in the beginning of the 20th century when Dehn formulated problems for finitely presented groups (word, conjugacy, isomorphism problems) that asked whether there even was an algorithmic solution. Later, Novikov proved the unsolvability of the word problem, and then came the proofs that there is no algorithm to decide whether a finitely presented group is trivial, finite, abelian, etc. Then, in the early "computer age", researchers started to use computers looking for special kinds of groups, or calculating structures of certain groups. However, the major impetus was given by the 1967 Oxford conference "Computational Problems in Abstract Algebra", where many of the fundamental methods in modern computational group theory were first presented; among those, the still-fundamental algorithm for permutation groups, namely, Sims's method for computing point stabilizer chains. In the next decade along with other new algorithms and applicational successes came the need for a machinery that could make programming easier[1]. In Aachen, Germany,

[1]For a more detailed history, see Neubüser's *An Invitation to Computational Group Theory* [Neu]

under the direction of Neubüser and in Sydney, Australia, led by Cannon, systems, with different distribution and "transparency" policies, but similar contents, started to evolve, now known as GAP and Magma, respectively.

Concrete computational results included the proof of the existence of the group called the "Baby Monster" by Sims, the proof of existence of Janko's J_4 by Norton, and the construction of the Cambridge "Atlas of Finite Groups".

Polynomial-Time Algorithms for Permutation Groups

The serious study of computational-complexity issues in group-theoretic computation began around 1980, inspired originally by applications to the classical problem of testing graph isomorphism [Lu1]. This required an evaluation and extension of a polynomial-time library for permutation-group computation.

It was observed that Sims's method for testing membership could be implemented in polynomial time [FHL], specifically, in time $O(sn^2 + n^6)$, where s is the number of generators describing the input group and n is the size of the permutation domain. Subsequently, Knuth in [Kn] gave an organization of the algorithm that he proved to run in time $\Theta(n^5 + sn^2)$. Jerrum in [Je] used a different data structure to achieve the same asymptotic time complexity.

Sims's algorithm uses a data structure that is based upon a specified chain of subgroups. It is set up as follows.

```
Given $G \leq Sym(\Omega)$, $G = G_0 > G_1 > \cdots > G_t = 1$
    for $i = 0$ to $t - 1$
        find a right transversal for $G_{i+1}$ in $G_i$
        find (Schreier) generators for $G_{i+1}$
```

For membership testing and other problems this data structure often enables transferring the problem from G_i to G_{i+1} (see Chapter II).

The problem with settitng up the data structure is that the number of generators possibly increases by a factor of $n - i$ at the ith iteration of the loop if we use point stabilizer series (G_i is the subgroup fixing the first i points) and we just naively apply Schreier's lemma (see Chapter II). If we just want to establish polynomial time, it is enough to observe that the number of generators for any group G permuting n elements can be kept under n^2 by the following process: Let S be a generating set for G. While there are $g, h \in S \setminus G_1$ such that $gh^{-1} \in G_1$ then replace h with gh^{-1} and discard duplicates. After this, there will be at most one generator for each coset of G_1, in addition to elements of G_1 (which do not necessarily generate G_1). Repeat the process for $S \cap G_1$ with respect to G_2, etc. At the end, we will have at most $|G_0 : G_1| + |G_1 : G_2| + \ldots + |G_{n-2} : G_{n-1}| \leq n(n+1)/2$ elements, that still generate G.

With a more careful organization, Knuth shows that the number of generators can be kept at $O(n)$. This is crucial to his $\Theta(n^5 + sn^2)$ time bound, which he goes on to show is the best possible utilizing point stabilizer chains (see [Kn]).

Another approach for setting up a data structure for membership testing (and computing the size of the group) [BLS1], using a different kind of subgroup chain achieves an $O(n^4 \log^c n + sn^2)$ time, and an even better asymptotic time bound of $O(sn^3 \log^c n)$ is given in [BLS2]. However, to achieve this the authors employ a very complicated algorithm, and use the classification of the finite simple groups. In [LRW2] for the special case of nilpotent groups an $O(n^4 + sn^2)$ method is mentioned which uses yet another kind of subgroup chain. This latter bound is further improved in this dissertation to $O(sn^3)$, using an easily programmable algorithm that performs very well in practice, too. (To see that this is an improvement, we note that the size of the input is sn.)

There is a large number of problems that have polynomial-time solutions, many of them are described in [Lu2] and [KL]. Some of the basic ones are: finding orbits of a group; finding minimal blocks of imprimitivity for a transitive group, therefore testing primitivity; finding the order of a group; testing membership in a group. As a consequence of membership-testing, there are polynomial time algorithms for: finding out whether a group is a subgroup of another; finding normal closures of subgroups; finding the commutator subgroup; testing solvability; testing nilpotence.

There are problems that are not known to have a polynomial-time solution in the general case, but we might have one in some special cases. A good example of this is the normalizer problem: given $G, H < K < Sym(\Omega)$, find $N_G(H)$, the normalizer of H in G. This problem has no known polynomial-time solution in the general case but, for nilpotent K, there is a polynomial-time solution in [KL], with no apparent attempt to make it practical. In [LRW2] (and in this dissertation) there is a version that was programmed and, in practice solved cases, with which the built-in methods for both Magma and GAP, that are based on backtracking, were not able to deal. The normalizer problem is known to be in polinomial time for solvable K, too.

Another example is the centralizer problem: given $G, H < Sym(\Omega)$, find $C_G(H)$, the centralizer of H in G. Polynomial-time solutions are known if G normalizes H, or when G is in the class Γ_d, that includes all solvable groups.

This dissertation shows novel data structures for the special cases of nilpotent and solvable groups. Using these data structures, some algorithms that were proved to have polynomial-time asymptotic behavior, can be turned into efficient computer programs. The usefulness of these data structures comes from finding natural normal series for the groups and natural maps from permutations to the vector spaces corresponding to the elementary abelian factors in these series.

The time necessary for the computation of these data structures is not negligible, even though it is less than setting up the usual point-stabilizer chain. This is not an issue if the input is known be nilpotent or solvable, respectively. However, as with any algorithm, that applies only to a special case but is more efficient in such a case, one should consider the cost of testing whether the input has the required properties. With this in mind, it is important to have very fast ways to test for the properties.

One of the main results of this dissertation is the presentation of such tests. Our algorithm for testing nilpotence of a group is extremely fast, both asymptotically ($O(sn \log n \log^* n)$) and in practice. The test for solvability is not much worse. Its asymptotic running time is dominated by finding a block system for a transitive group, so it runs in $O(sn^2)$ time; the rest of the algorithm runs in $O(sn \log^3 n)$ time. This test is also very fast in practice. Note that the input size is $\Theta(sn)$, so testing nilpotence is "almost linear" (see [BCFS],[Se]).

To illustrate the utility of the data structures, we show some algorithms that make good use of them. For nilpotent groups, algorithms for three problems are shown: if G and H are subgroups of a nilpotent group, we find $N_G(H)$, $N \cap H$ and $C_G(H)$. These were known to be in polynomial time [Lu3] [KL], our data structure facilitates programs for them that perform well in practice.

For solvable groups, we illustrate the usefulness of the data structure by showing how to reduce finding Sylow p-subgroups of a solvable group to solving a series of linear equations. For permutation groups, this problem has long been known to be in polynomial time, even for non-solvable groups (see [Ka], wich also has a simplified algorithm for solvable groups). In [KT] a different algorithm is given for solvable groups. An NC (parallel) algorithm is given for the solvable case in [KLM]. In a more general setting an algorithm requiring a polynomial number of group and field operations is presented in [EW]. Our algorithm has a similar abstract structure to that of [EW] and is a variation of the one in [KLM], making use of the vector spaces that naturally arise from the permutation structure and are part of our data structure.

The structure of this dissertation is the following. In Chapter II we summarize the notation and recall basic definitions used throughout the whole dissertation. In Chapters III to V we provide theoretical background for and show how to build a data structure for permutation p-groups that represents more structural information about the group than the usual point-stabilizer chain. Furthermore we show that building our data structure costs less asymptotically than computing the point-stabilizer chain. In Chapter VI we describe a very fast method for recognizing whether a permutation group is nilpotent. This algorithm utilizes a p-group test, which is also given. In Chapter VII we present some algorithms the implementation of which utilizes the data structure described in Chapter V. The last three chapters deal with solvable permutation groups. We start with a description of the structure of solvable permutation groups in Chapter VIII, then we describe a data structure that corresponds to a normal series with elementary abelian factors, and provide means to treat the factors as vector spaces enabling the use of linear algebra in computations. In Chapter IX we show how to build that data structure. In Chapter X we present an algorithm for recognizing solvability of a permutation group and then we illustrate the usefulness of the data structure in an algorithm for computing Sylow p-subgroups.

CHAPTER II

NOTATION AND BACKGROUND

Basics

Let Ω be a set. We denote the group of all permutations on Ω by $Sym(\Omega)$. If $\Omega = \{0, 1, \ldots, n-1\}$ then we write $Sym(n)$ instead of $Sym(\Omega)$. Let $G \leq Sym(\Omega)$. We denote the elements of G by small Roman letters, the elements of Ω by small Greek letters, and if the permutation g takes ω to τ then we write $\tau = \omega^g$. Sets of permutations, including subgroups of $Sym(\Omega)$, will be denoted by capital letters. Products of permutations are written left to right, i.e. $\omega^{gh} = (\omega^g)^h$.

If G is a group, Ω is a set, and there is a homomorphism $\theta : G \to Sym(\Omega)$ then we say that G is *acting on* Ω. If the kernel of this homomorphism consists of the identity element of G then we call the action *faithful.* If $G \leq Sym(\Omega)$ is acting on the set Δ and the homomorphism involved is obvious, we denote the image of G in $Sym(\Delta)$ by G^Δ.

Let $A \subseteq Sym(\Omega)$ be a set of permutations and $\omega \in \Omega$, then we use the notation ω^A to denote the set $\{\omega^g : g \in A\}$. The group $G \leq Sym(\Omega)$ is called *transitive* iff $\omega^G = \Omega$. For a nontransitive group G, the set $\{\omega^G : \omega \in \Omega\}$ is a partition of Ω into *orbits* of G. On each orbit Δ, G acts naturally (the mentioned homomorphism being the one that maps a permutation to its restriction to Δ), this action is transitive and is called a *transitive constituent* of G, denoted by G^Δ in accordance with the notation used for actions. The kernel of this action is the subgroup of G that pointwise fixes the orbit.

Let $\Delta \subseteq \Omega$, let G be a group acting on Ω and let $g \in G$. We denote $\Delta^g = \{\delta^g : \delta \in \Delta\}$. Let G be transitive on Ω and let $\Delta \subset \Omega$. If $\Delta \neq \emptyset$ and, for all $g \in G$, either $\Delta^g = \Delta$ or $\Delta^g \cap \Delta = \emptyset$ then we call Δ a *block (of imprimitivity)*. In this case, the set $\{\Delta^g : g \in G\}$ is a partition of Ω into blocks, and is called a *block system*. *Trivial block systems* are $\{\Omega\}$ and $\{\{\omega\} : \omega \in \Omega\}$, other block systems are called *nontrivial*. If $\boldsymbol{\Delta}$ is a block system, G acts on $\boldsymbol{\Delta}$. The kernel of this action is the subgroup of G that setwise stabilizes the blocks. If a group has no nontrivial block systems, it is called *primitive*.

By $\langle S \rangle$ we will denote the group generated by the elements in S, where $S \subset G$. In the following, when we say "given a group...", we always mean it is given by a set of generators.

We will use the usual notation $H < G$ to denote that H is a subgroup of G. We write $H \leq G$ if H is not necessarily a proper subgroup. We denote the fact that H is a (not necessarily proper) normal subgroup of G by $H \triangleleft G$. The trivial group is denoted by the 1.

Let $H \leq G$, then we call a complete set of right coset representatives, (i.e. a set T for which $\bigcup_{t \in T} Ht = G$ and $\forall t_1, t_2 \in T, t_1 \neq t_2 : Ht_1 \cap Ht_2 = \emptyset$), a *right transversal (for H in G)*.

A construction of Schreier plays a central role in computational group theory. It produces generators for a subgroup. If the subgroup is of moderate index, for which we have an efficient method to recognize membership (such as point stabilizer subgroups of permutation groups), the method is useful in algorithms.

Lemma 2.1 (Schreier)

Let G be a group, $H < G$ a subgroup, and let T be a right transversal for H in G. Let S be a generating set for G. Then $H = \langle \{t_1 s t_2^{-1} : t_1, t_2 \in T, s \in S\} \cap H \rangle$ $\square$

Note that for any (t_1, s) pair there exists exactly one t_2 such that $t_1 s t_2^{-1} \in H$, so the number of the Schreier generators does not exceed $|S||G : H|$ (there might be repetitions).

The permutations in $G < Sym(\Omega)$ that fix $\omega \in \Omega$ form a subgroup, which we call the *point stabilizer subgroup (of ω in G)* and denote by G_ω. For the subgroup of G that fixes all the points in $\Delta \subseteq \Omega$ we use the notation G_Δ, and call it the *pointwise set stabilizer (of Δ in G)*, while for the subgroup that stabilizes Δ as a set only, i.e. $\{g \in G : \forall \delta \in \Delta, \delta^g \in \Delta\}$ we use the name *set stabilizer (of Δ in G)*, and the notation $G_{\{\Delta\}}$. If $\Delta = \{\delta\}$, we use the notation G_δ for G_Δ. Trivially $G_\Delta \leq G_{\{\Delta\}}$.

A fundamental structure in computational permutation group theory is the point stabilizer chain. Let $\Omega = \{\omega_1, \ldots, \omega_n\}$, then the series $G = G^{(0)} \geq G^{(1)} \geq \ldots \geq G^{(n-1)} = 1$ is called a point stabilizer chain with respect to the sequence $(\omega_1, \ldots, \omega_n)$ if $G^{(i)} = G^{(i-1)}_{\omega_i}$. This makes it possible to solve the most natural problem in computational group theory:

Membership problem.

Given a group by a set of generators, $G = \langle S \rangle$ and an element of a supergroup of G (in the permutation group setting an element of $Sym(\Omega)$). Is $g \in G$? $\square$

To see how a series of subgroups helps in answering the above question, we introduce the following:

Lemma 2.2

Let K be a group, $H < G \leq K$, let T be a right transversal for H in G and let $\phi : K \to T$ a (computable) map such that $\forall g \in G, g\phi(g)^{-1} \in H$. Let us assume, furthermore, that we can decide membership in H. Then we can decide also whether an element of K is an element of G.

Proof

Let $g \in K$. Then $g \in G$ iff $g\phi(g)^{-1} \in H$. □

One can make use of this lemma using a subgroup chain $G = G_0 > G_1 > \cdots > G_t = 1$, with transversals for the subsequent groups in the chain to test membership in G. It is costumary to refer to this procedure as *sifting.*

In the permutation group setting, we can get all the ingredients of the above lemma for any two successive elements of the chain, so we can use sifting to test membership. The base case is a test of whether a given permutation is the identity. The ingredients that we are looking for are a right transversal T, and a mapping ϕ with the required property. Let $L \leq K$ and $\omega \in \Omega$. We want a right transversal for L_ω in L. A useful property of point stabilizer subgroups is that for any two elements g and h of L, g and h are in the same coset of L_ω iff $\omega^g = \omega^h$, since this is equivalent with $\omega^{gh^{-1}} = \omega$, i.e. $gh^{-1} \in L_\omega$. So, for T, we only need to find elements t_τ of L that take ω to τ, for all $\tau \in \omega^L$. This can be done easily in conjunction with a naive transitive closure algorithm that computes the orbit of ω. While doing this, the computation of ϕ is an easy matter, if $\omega^g \in \omega^L$, let $\phi(g) = t_{\omega^g}$, otherwise, let $\phi(g) = t_\omega$.

Having a subgroup chain with the transversals also makes it easy to compute the size of the group, for given $H < G$ and a right transversal T of H in G, $|G| =$

$|H||G:H| = |H||T|$.

Special Cases

Not surprisingly, if we restrict the domain of our interest, we may find that there exist polynomial-time solutions for problems that are not necessarily polynomial in the wider domain, or faster methods for special cases even when the general case is polynomial. A natural restriction of the permutation group domain could be the investigation of solvable groups, or the more restricted domain of nilpotent groups, which are direct products of p-groups.

We recall the definitions of the above mentioned classes of groups. Let G be a group, $g, h \in G$. The *commutator of g and h* is defined to be the product $g^{-1}h^{-1}gh$, and is denoted by $[g,h]$. If H is a subgroup of G, we denote by $[G,H] = \langle\{[g,h] : g \in G, h \in H\}\rangle$. We call $G' = [G,G]$ the derived group of G. If the series $G \geq G' \geq G'' \geq \ldots$ stabilizes at the trivial group, we call the group G *solvable*. The series $G = L^0(G) \geq L^1(G) \geq L^2(G) \geq \ldots$, where $L^i(G) = [G, L^{i-1}(G)]$ for $i > 0$, is called the *lower central series of G*. If it stabilizes in the trivial group, G is called *nilpotent*. (Other, equivalent, definitions of both classes may be found in group theory texts.) It is not hard to see that if $G = \langle S\rangle > H = \langle U\rangle$ then $[G,H] = \langle\{[s,u] : s \in S, u \in U\}\rangle^G$, where for $X \leq G$, X^G denotes the *normal closure of X in G*, i.e. the unique smallest normal subgroup of G that contains X.

If $G, H \leq Sym(\Omega)$, we denote $N_G(H) = \{g \in G : \forall h \in H, g^{-1}hg \in H\}$ and call it the *normalizer of H in G*, similarly, we denote $C_G(H) = \{g \in G : \forall h \in H, gh = hg\}$, and call it the *centralizer of H in G*. If $N_G(H) = G$, we say that G *normalizes* H, if $C_G(H) = G$, we say that G *centralizes* H. $Z(G) = C_G(G)$ is called the *center of G*.

CHAPTER III

THE STRUCTURE OF PERMUTATION p-GROUPS

In this chapter, we will see a description of the Sylow p-subgroups of $Sym(n)$, the symmetric group acting on the set $\{0, \dots, n-1\}$. In the whole chapter, p is a fixed prime number. Since every p-subgroup of $Sym(n)$ is a subgroup of some Sylow p-subgroup, and the Sylow p-subgroups are conjugates of each other, i.e. they can be obtained from one another by relabeling points, it is sufficient to exhibit the structure of one of them.

The structure of Sylow subgroups of the symmetric groups is well known. In [Ha] (pp. 81-83) there is a discussion in terms of wreath products.

Definition 3.1

Let H and G be permutation groups on sets Ω and $\Sigma = \{\sigma_1, \dots, \sigma_t\}$ respectively. For $h_1, \dots, h_t \in H$ and $g \in G$ we define $f = (h_1, \dots, h_t, g)$ as a permutation of $\Omega \times \Sigma$ by $(\omega, \sigma_i)^f = (\omega^{h_i}, \sigma_i^g)$. Then the wreath product of H and G, written as x1$H \wr G$, is defined as the group of permutations $\{(h_1, \dots, h_t, g) : h_1, \dots, h_t \in H, g \in G\}$. □

The wreath product is associative in the sense that for $G \leq Sym(\Omega)$, $H \leq Sym(\Sigma)$ and $K \leq Sym(\Theta)$, $(G \wr H) \wr K$ is isomorphic to $G \wr (H \wr K)$ and if we identify the sets $(\Omega \times \Sigma) \times \Theta$ and $\Omega \times (\Sigma \times \Theta)$, in the natural way, with $\Omega \times \Sigma \times \Theta$, then they are identical. If we denote the cyclic group of order p by P, then the Sylow

p-subgroups of the symmetric group on a set of p^k elements can be identified with $P \wr P \wr \cdots \wr P$, where there are k factors.

We give a description here that is logically equivalent with the above but is in terms of the possible transformations of a series of mechanical structures (toys). The toys are devices on which one can move around distinctly marked dots in a controlled way. After finishing each movement, the each dot stops at one of the numbered positions of the toy. A permutation corresponds to each possible transformation (movement) that takes the number i to j iff the dot that before the move occupied position i, is moved to position j. For each k we construct a toy D_k and a set of permutations $\{\tau_1, \ldots, \tau_k\}$ that will belong to elementary movements of the device, and we will show that these permutations generate a Sylow p-subgroup of $Sym(p^k)$ by showing that the number of possible positions of the toy equals the size of that Sylow subgroup. We will call this Sylow subgroup the canonical Sylow p-subgroup $P^{(k)}$ of $Sym(p^k)$ and we will call the generators $\{\tau_1, \ldots, \tau_k\}$ the canonical generating set for $P^{(k)}$.

The toy D_1 is a disk that has p dots on it, arranged at vertices of a regular p-gon. The disk can be turned around an axle at its center and has stop positions at angles divisible by $360/p$ degrees, so it can assume p different positions. The positions are numbered $0, 1, \ldots, p-1$, clockwise in cyclic order. The permutation τ_1 associated with D_1 is $(0, 1, \ldots, p-1)$, this corresponds to the turn of the disk clockwise by $360/p$ degrees.

D_i, for $i \geq 2$ is disk that can be moved the same way, but at the vertices of the regular p-gon, instead of dots, it has a copy of a D_{i-1} toy attached. So it has disks of decreasing sizes, each of which can be turned into p different positions. The turn of a disk of course moves all the smaller disks attached to it. The smallest disks have the dots on them (p^i dots altogether) and the positions are numbered using the following scheme. The positions of dots belonging to one of the second biggest disks are numbered according to the numbering of D_{i-1}. The numbering of the positions belonging to the other disks are derived from this as follows. We assign the number $jp^{i-1}+m$ to the position where the dot covering position m would end up if we turned the biggest disk by $360j/p$ degrees clockwise, not changing the relative positions of the attached smaller disks ($0 \leq i < p$). The permutations associated with D_i are $\{\tau_1, \ldots, \tau_i\}$, where $j^{\tau_i} = j + p^{i-1}$, where the addition is modulo p^i. Sometimes we will refer to i as the *size* of the biggest disk of D_i. A D_2 toy for $p = 3$ is shown on Figure 1.

We call a series of movements of the disks that are permitted by the constraints, a transformation of the structure. These transformations correspond to permutations of $Sym(p^k)$, and these permutations obviously constitute a subgroup. The size of this subgroup is the number of different configurations, i.e the number of possible positions of the p^i dots. Let us denote this number by m_i. It is easy to find a recursive formula for m_i. For $i = 1$, the description of D_1 shows that $m_1 = p$. For $i \geq 2$, we can turn the outermost wheel into p different positions, and in each position we can turn each of the p copies of D_{i-1} into m_{i-1} different positions independently, so $m_i = pm_{i-1}^p$. The solution of this recursion is $m_i = p^{\frac{p^i-1}{p-1}}$. This holds for $i = 1$, and it is also straightforward, that $p(p^{\frac{p^{i-1}-1}{p-1}})^p = p^{\frac{p^{i-1}-1}{p-1}p+1} = p^{\frac{p^i-1}{p-1}}$, since $\frac{p^{i-1}-1}{p-1}p+1 = \frac{p^i-p+p-1}{p-1} = \frac{p^i-1}{p-1}$.

This computation shows us that the group corresponding to all possible positions of the numbers on D_i is a Sylow p-subgroup of $Sym(p^i)$, for the largest power of p

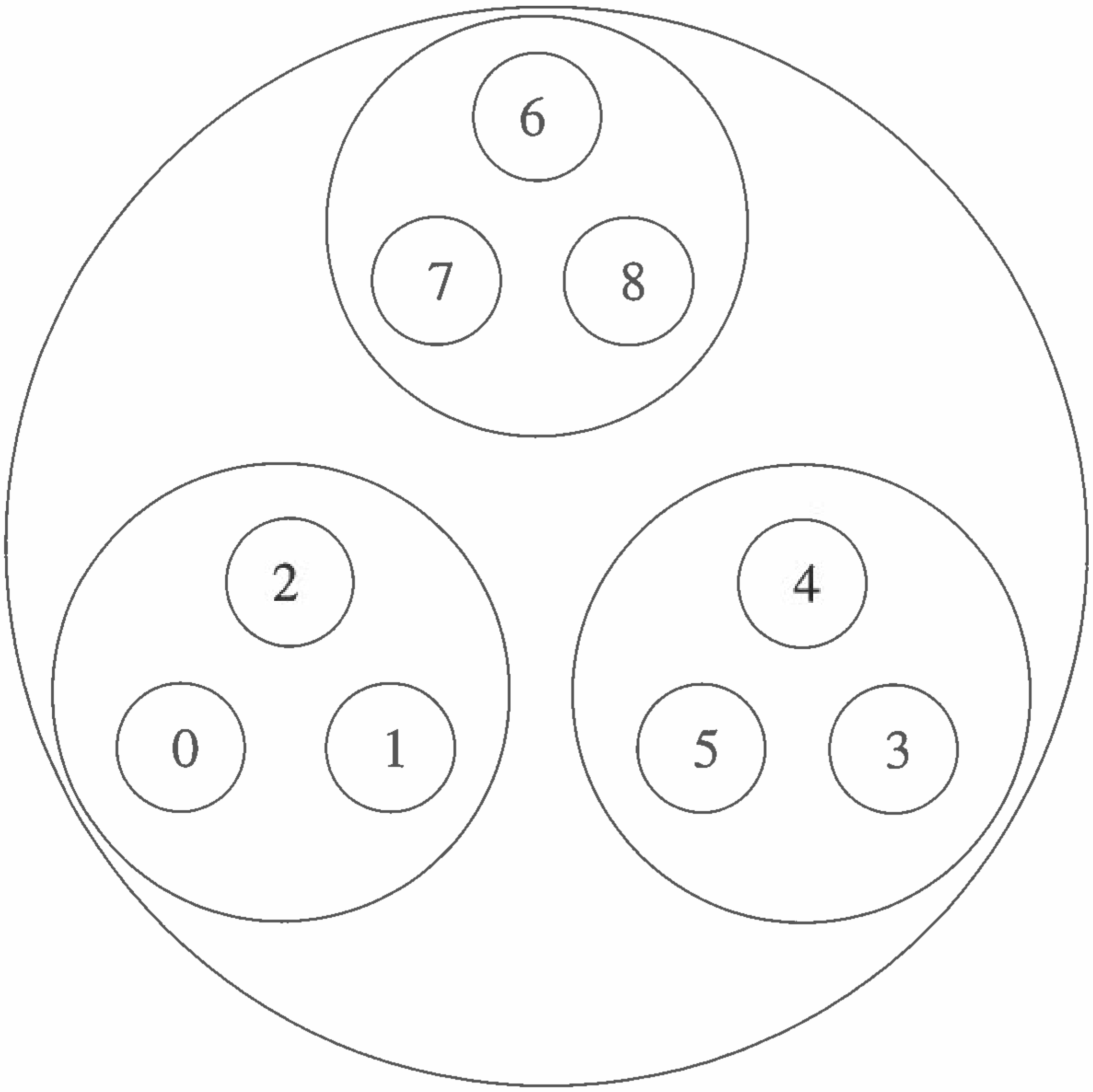

Figure 1: A D_2 toy for $p = 3$

in $p^i!$ is exactly m_i. (The exponent of p in the prime power decomposition of $n!$ is $\lfloor n/p \rfloor + \lfloor n/p^2 \rfloor + \cdots + \lfloor n/p^k \rfloor$, where k is such that $p^{k+1} > n$.)

The toy D_i by its construction immediately reveals the block structure of the Sylow p-subgroups of $Sym(p^i)$. The points corresponding to any disk on D_i are necessarily moving together, constituting blocks. The disks with the same size constitute block systems, these are organized to a hierarchy by the sizes of the disks. This hierarchy can be depicted as a p-ary tree in which every node corresponds to a disk (block) and each node is connected to those corresponding to maximal subblocks of it. This tree is usually called a structure tree for the group. For p-groups, if we impose a cyclic order on the nodes with common parent node, the group becomes a subgroup of the automorphism group of its structure tree. In the case of a Sylow p-subgroup of $Sym(p^i)$ we have the full automorphism group, since in this case the tree and D_i are functionally equivalent (think of the axels of the disks in D_i as the nodes in the tree, the same cyclic order is imposed on the substructures in both cases).

It is also easy to see that the permutations $\{\tau_1, \ldots, \tau_i\}$ constitute a generating system for the group corresponding to D_i. This is trivially true for D_1. Now suppose that all the possible movements of D_{i-1} can be achieved using the elementary movements corresponding to $\{\tau_1, \ldots, \tau_{i-1}\}$. Then we can achieve every movement of D_i by first moving each of the size $i-1$ disks in the position of the first one turning the biggest disk (this corresponds to a power of τ_i), then rearrange the smaller disks in that position using the inductive hypothesis. After we did this for all size $i-1$ disks, we can turn the biggest disk into its desired position, again using a move corresponding to a power of τ_i.

Normal Structure

From the theory of p-groups it is well known that they have normal series with factors of size p, which is necessarily a chief series (see [Ha]). Here, and in the next chapter we will see a constructive proof of this, using the permutation action (in particular, block structure) of the group, to obtain a normal series that corresponds to the block hierarchy, and then further refining it to get the whole chief series. It is enough to show this for a Sylow p-subgroup of $Sym(p^k)$, since every p-group can be embedded into direct products of such groups, since every transitive constituent of a p-group is acting on a p-power number of points. For direct products, it is easy to obtain a chief series from the chief series of the direct factors. Finally, if H is a p-group and we have a chief series $H = H_0 > H_1 > \cdots > H_m = 1$ for H and $G < H$, then, after removing duplicates $G \cap H_0 \geq G \cap H_1 \geq \ldots \geq G \cap H_m = 1$ is a chief series for G.

Let now $P = P^{(k)}$ be the canonical Sylow p-subgroup of $Sym(p^k)$. We will call the subgroup, that fixes all of the disks of size j (and therefore all of the disks of bigger sizes) on the corresponding D_k, a *level stabilizer*. It is clear that a level stabilizer is normal in P, because it is the kernel of the homomorphism that maps the group corresponding to D_k to the one corresponding to D_{k-j} by identifying the size j disks with the dot the number of which is $1/p^j$ times the smallest number belonging to the disk.

Let us denote the series of the level stabilizers of $P = P^{(k)}$ by $P = P_k > P_{k-1} > \ldots > P_0 = 1$. Then the factors $V_i = P_i/P_{i-1}$, $i = 0, \ldots, k-1$ are elementary abelian p-groups, since they correspond to movements of the size i disks on D_k, while the bigger size disks are fixed. Since P contains the permutations corresponding to every possible movement of the disks, and there are p^{k-i} disks of size i, the size of V_i is $p^{p^{k-i}}$. Since V_i is elementary abelian, it can be viewed as a vector space over $GF(p)$, the basis vectors being the permutations corresponding to transformations that rotate one of the size i disks clockwise by $360/p$ degrees and fix all the other disks of size i. P-normal subgroups that lie between P_i and P_{i-1} correspond to P-invariant subspaces of V_i, where the action of $g \in P$ on V is defined by $(hP_{i+1})^g = h^g P_{i+1}$. It is easy to see that this action is simply a permutation of the coordinates. To show this, we have to show that all elements of a generating system have this kind of action. For the canonical generating system, it is trivial: the generators $\tau_1, \ldots, \tau_{i+1} \in P_{i+1}$, τ_i turns the size i disks not changing their positions, so conjugating by it fixes the basis vectors. The rest of the generators, moving whole blocks, do not interfere with with the amount of rotation on each disk on the ith level either, so they move each basis vector to another. In the next chapter we will see a construction of P-invariant subspaces of $GF(p)^{p^k}$, via a basis that is easily computable and has additional useful properties.

CHAPTER IV

INVARIANT SUBSPACES OF $GF(p)^{p^k}$

In Chapter III we saw that in order to refine the normal series corresponding to the level stabilizer subgroups of P to a chief series, we need to find invariant subspaces of $V = V^{(k)} = GF(p)^{p^k}$ under the action of $P^{(k)}$. We also saw that $P^{(k)}$ is acting on this vector space as a group of linear transformations that permute the coordinates of vectors written in a natural basis. We will denote this basis by $\mathbf{e}_0, \mathbf{e}_1, \ldots, \mathbf{e}_{p^k-1}$. Our goal is now to find a series of subspaces $V = V_0 > V_1 > \cdots > V_{p^k} = 1$ and a basis $\mathbf{b}_0, \mathbf{b}_1, \ldots, \mathbf{b}_{p^k-1}$ such that $V_i = Span(\mathbf{b}_i, \mathbf{b}_{i+1}, \ldots, \mathbf{b}_{p^k-1})$ and V_i is invariant under the action of P. In what follows, we identify the elements of $GF(p)$ by the integers modulo p. We define a series of linear transformations of $GF(p)^{p^k}$, $T_1, T_2, \ldots, T_k$ by

$$\mathbf{e}_m T_j = \mathbf{e}_{m^{\tau_j}} \text{ for } j = 1, \ldots, k,\ m = 0, \ldots, p^k - 1.$$

Here $\{\tau_1, \ldots, \tau_k\}$ is the canonical generating set for $P^{(k)}$, defined in Chapter III. These transformations directly correspond to the action of the canonical generating system of P on V. In the following proposition we will denote the dot product of two vectors $\mathbf{u}$ and $\mathbf{v}$ by $\langle \mathbf{u}, \mathbf{v} \rangle$.

Proposition 4.1

There exists a p^k by p^k matrix $B^{(k)}$ such that, if we define $\mathbf{b}_i = \sum_{j=0}^{p^k-1} B^{(k)}_{i,j} \mathbf{e}_j$ and $V_i = Span(\mathbf{b}_i, \mathbf{b}_{i+1}, \ldots, \mathbf{b}_{p^k-1})$ for $i = 0, 1, \ldots, p^k - 1$ then a.) V_i is invariant under T_j, for $1 \leq j \leq k$, $0 \leq i < p^k$,

b.) $\langle \mathbf{v}_i, \mathbf{v}_j \rangle = 0$ for $\mathbf{v}_i \in V_i$, $\mathbf{v}_j \in V_j$ where $i + j \geq p^k$, but

$\langle \mathbf{v}_i, \mathbf{v}_j \rangle \neq 0$ for $\mathbf{v}_i \in V_i \backslash V_{i+1}$, $\mathbf{v}_j \in V_j \backslash V_{j+1}$ if $i + j = p^k - 1$,

c.) $B^{(k)^{-1}} = B^{(k)}$.

Furthermore, V_i is the unique $p^k - i$-dimensional subspace that has the property in a.) .

Proof

The proof is a construction of such a matrix. We do the construction recursively.

For $k = 1$ let $\mathbf{b}_0 = \mathbf{e}_0$ and let $\mathbf{b}_i = \mathbf{b}_{i-1} - T_1\mathbf{b}_{i-1}$ for $i = 1, ..., p-1$. This is obviously a basis. Let the ith row of $B = B^{(1)}$ be the coordinates of $\mathbf{b}_i$ in the natural basis. In the following we show why the properties a.)-c.) are true.

a.) $V_i = Span(\mathbf{b}_i, ..., \mathbf{b}_{p-1})$ is invariant under T_1.

It is easy to see that $\mathbf{b}_i[j] \equiv ((-1)^j \binom{i}{j}) \bmod p$. Since for p prime

$$\binom{p-1}{j} \equiv \frac{(p-1)(p-2)\cdots(p-j)}{1 \cdot 2 \cdots j} \equiv (-1)^j \bmod p,$$

$\mathbf{b}_{p-1} = (1, 1, \ldots, 1)$. So for $i = p - 1$, the statement is trivially true. Suppose that for $j > i$ V_j is invariant under T_1. Then we show that V_i is invariant under T_1 as well. It is enough to show that $T_1\mathbf{b}_i \in V_i$. But this follows from the fact that $\mathbf{b}_i - T_1\mathbf{b}_i = \mathbf{b}_{i+1} \in V_{i+1} \subset V_i$.

b.) About the dot products.

It is enough to show that the statement holds for the **b**'s in place of the **v**'s . So let $i + j \geq p$, we want to show that $\langle \mathbf{b}_i, \mathbf{b}_j \rangle = 0$.

$$\langle \mathbf{b}_i, \mathbf{b}_j \rangle \equiv \sum_{l=0}^{p-1} (-1)^l \binom{i}{l} (-1)^l \binom{j}{l} = (\sum_{l=0}^{p-1} \binom{i}{l} \binom{j}{j-l}) =$$

$$\binom{i+j}{j} = \frac{(i+j)!}{i!j!} \equiv 0 \bmod p$$

since $i < p \leq i + j$ and $j < p$ and so there is a factor of p in the numerator, but there is none in the denominator. (For the previous well-known equality one can imagine how many ways one can choose j balls from a basket of i red and j blue balls, all distinguishable from each other. The terms in the sum stand for the cases where l red and $j - l$ blue balls were chosen.)

If $i + j = p - 1$ then $\langle \mathbf{b}_i, \mathbf{b}_j \rangle = \binom{p-1}{j} \bmod p = (-1)^j$.

c.) $B := B^{(1)} = B^{(1)^{-1}}$

This is equivalent to $B^2 = I$. We use $B_{i,j}$ for the jth coordinate of $\mathbf{b}_i$.

$$B^2_{i,j} = \sum_{l=0}^{p-1} B_{i,l} B_{l,j} = \sum_{l=0}^{p-1} (-1)^l \binom{i}{l} (-1)^j \binom{l}{j} = \sum_{l=0}^{p-1} (-1)^{l-j} \binom{i}{j} \binom{i-j}{l-j}$$

(Consider two methods of counting how many ways one can paint j balls red and $l - j$ balls blue out of i distinguishable white balls. The first method is to select the l balls to be painted first and then select j out of the l to be painted red, the second method is to select the balls to be painted red and then the balls to be painted blue. Also note that $(-1)^{l+j} = (-1)^{l-j}$.) So

$$B^2_{i,j} = \binom{i}{j} \sum_{l=0}^{p-1} (-1)^{l-j} \binom{i-j}{l-j}$$

Now the sum is 1 if $i = j$, 0 if $i < j$ because in this case each term is 0, and it is just another expession for $(1-1)^{i-j}$ if $i > j$

So $B^2_{i,j} = \binom{i}{j}\delta_{i,j} = \delta_{i,j}$, where δ is the Kronecker symbol, which shows that $B^2 = I$.

V_i's are unique.

Suppose W is a T_1 invariant subspace of $V = GF(p)^p$. Since $\mathbf{b}_0, \cdots, \mathbf{b}_{p-1}$ constitute a basis for V, each vector in W is expressible as $w_0\mathbf{b}_0 + \ldots + w_{p-1}\mathbf{b}_{p-1}$. Let i be the smallest index, for which there exist $\mathbf{w} \in W$ such that $w_i \neq 0$. Clearly $\dim(W) \leq p - i$. We show that W must contain $\mathbf{b}_i, \ldots, \mathbf{b}_{p-1}$, so $W \supset V_i$, and since V_i is a $p-i$ dimensional subspace of V, it follows that $V_i = W$.

$$\begin{array}{llll}
\mathbf{w}_i \overset{\text{def}}{=} & \mathbf{w} & = w_i\mathbf{b}_i + \cdots + w_{p-1}\mathbf{b}_{p-1} & \in W, \text{ so} \\
\mathbf{w}_{i+1} \overset{\text{def}}{=} & \mathbf{w}_i - T_1\mathbf{w}_i & = w_i\mathbf{b}_{i+1} + \cdots + w_{p-2}\mathbf{b}_{p-1} & \in W, \text{ so} \\
\vdots & & & \\
\mathbf{w}_{p-2} \overset{\text{def}}{=} & \mathbf{w}_{p-3} - T_1\mathbf{w}_{p-3} & = w_i\mathbf{b}_{p-2} + w_{i+1}\mathbf{b}_{p-1} & \in W, \text{ so} \\
\mathbf{w}_{p-1} \overset{\text{def}}{=} & \mathbf{w}_{p-2} - T_1\mathbf{w}_{p-2} & = w_i\mathbf{b}_{p-1} & \in W,
\end{array}$$

so $\mathbf{b}_{p-1} \in W$ (since $w_i \neq 0$) and from this and the last but first line above it follows that $\mathbf{b}_{p-2} \in W$, etc.

Remarks:

(i) In the proof of c.) it is not important that p is prime.

(ii) b.) remains true if we substitute p by some q^m where q is prime and we write mod$q$ whereever we had written mod$p$ in the proof.

(iii) The transformation $I - T_1$ takes each but the last basis vector to a higher indexed one (namely to the next one) and takes the last one to $\mathbf{0}$.

Now let us concentrate on the $k > 1$ case. The construction is the following. Suppose we already constructed (using this recursive procedure) the matrix with properties a.)-c.) for $GF(p)^{p^{k-1}}$ and the basis corresponding to it is $\mathrm{b}_0^{(k-1)}, \dots, \mathrm{b}_{p^{k-1}-1}^{(k-1)}$. We construct $\mathrm{b}_0^{(k)}, \dots, \mathrm{b}_{p^k-1}^{(k)}$ as follows:
if $p|i$ then let

$$\mathrm{b}_i^{(k)}[j] = \begin{cases} \mathrm{b}_{i/p}^{(k-1)}[j] & \text{if } j < p^{k-1} \\ 0 & \text{otherwise} \end{cases}$$

and if $p \not| i$ then let

$$\mathrm{b}_i^{(k)} = \mathrm{b}_{i-1}^{(k)} - T_k \mathrm{b}_{i-1}^{(k)}.$$

We observe that this construction is the equivalent to

$$\mathrm{b}_i^{(k)}[j] = \mathrm{b}_{\lfloor i/p \rfloor}^{(k-1)}[j \bmod p^{k-1}]\mathrm{b}_{i\,\mathrm{mod}\,p}^{(1)}[\lfloor j/p^{k-1} \rfloor],$$

because T_k moves around the coordinates, that have the same index modulo p, in a p-cycle. (in particular, the first p^{k-1} coordinates of each $\mathrm{b}_i^{(k)}$ vector constitute a copy of $\mathrm{b}_{\lfloor i/p \rfloor}^{(k-1)}$ and $T_k \mathrm{b}_{ip-1}^{(k)} = \mathrm{b}_{ip-1}^{(k)}$ for $i = 1, \dots, p^{k-1}$).

a.) By induction on k we show that for each of the transformations $T_1, \dots, T_k$, $I - T_j$ takes $\mathrm{b}_i^{(k)}$ to some $\mathrm{b}_l^{(k)}$, where $l > i$, or to $\mathbf{0}$. This is enough, since then $T_j \mathrm{b}_i^{(k)} \in V_i$ following the argument of the proof for the $k = 1$ case. The base case for the induction is the $k = 1$ case, for which the statement holds. Suppose it holds for $k-1$ and consider $\mathrm{b}_0^{(k)}, \ldots, \mathrm{b}_{p^k-1}^{(k)}$. $(I - T_k)\mathrm{b}_i^{(k)} = \mathrm{b}_{i+1}^{(k)}$ if $i+1 \not| p$ and $\mathbf{0}$ otherwise follows directly from the construction. For $j < k$ observe that $(I - T_j)\mathrm{b}_i^{(k)}$ is the same vector as $(I - T_j)\mathrm{b}_{\lfloor i/p \rfloor}^{(k-1)}$ augmented with $p^k - p^{k-1}$ 0 coordinates. Since by the inductive hypothesis $(I - T_j)\mathrm{b}_{\lfloor i/p \rfloor}^{(k-1)}$ is either $\mathbf{0}$ or some $\mathrm{b}_l^{(k-1)}$ for some $l > \lfloor i/p \rfloor$, the augmented vector is either $\mathbf{0}$ or some $\mathrm{b}_{lp}^{(k)}$, where $lp > i$.

For the proof of b.) and c.) we introduce another basis for $GF(p)^{p^k}$. Let $\mathrm{a}_0^{(k)} = \mathrm{e}_0$ and $\mathrm{a}_i^{(k)} = \mathrm{a}_{i-1}^{(k)} - S^{(k)}\mathrm{a}_{i-1}^{(k)}$, where $S^{(k)}(\mathrm{e}_i) = \mathrm{e}_{i+1}$, where $i+1$ is taken modulo p^k. It is not hard to see that $\mathrm{a}_i[j] = (-1)^j \binom{i}{j} \bmod p$. In fact $\mathrm{a}_i^{(1)} = \mathrm{b}_i^{(1)}$. Let $A_{i,j}^{(k)} = \mathrm{a}_i^{(k)}[j]$, then for $k > 1$ $A_{i,j}^{(k)} = A_{i \bmod p^{k-1}, j \bmod p^{k-1}}^{(k-1)} A_{\lfloor i/p^{k-1} \rfloor, \lfloor j/p^{k-1} \rfloor}^{(1)}$. To show this we prove by induction on k the last row of $A^{(k)}$ consists of all 1's. It is true for $k = 1$, since $A^{(1)} = B^{(1)}$. Suppose that the last row of $A^{(k-1)}$ consits of all 1's. The first p^{k-1} rows of $A^{(k)}$ are the rows of $A^{(k-1)}$ augmented by 0's. Therefore

$$A_{p^{k-1},j}^{(k)} = \begin{cases} 1 & \text{if } j = 0 \\ -1 & \text{if } j = p^{k-1} \\ 0 & \text{otherwise} \end{cases}$$

In the next p^{k-1} rows we will have a copy of $A^{(k-1)}$ then a copy of $-A^{(k-1)}$, followed by all 0's. In the next row we have:

$$A^{(k)}_{2p^{k-1},j} = \begin{cases} 1 & \text{if } j = 0 \\ -2 & \text{if } j = p^{k-1} \\ 1 & \text{if } j = 2p^{k-1} \\ 0 & \text{otherwise} \end{cases}$$

From this we can see the pattern. Each p^{k-1} by p^{k-1} block of $A^{(k)}$ is a copy of $A^{(k-1)}$ multiplied by a constant, which is in the upper left corner of it. The series of these constants equals a row of $A^{(1)}$. By the induction hypothesis, this constant will fill up the last row of the block, and the upper left corners of the next row of blocks will be filled up with the numbers from the next row of $A^{(1)}$. Since the last row of $A^{(1)}$ is all 1's, so is the last row of $A^{(k)}$. Now if $i = i_{k-1}p^{k-1} + \cdots + i_1 p + i_0$ and $j = j_{k-1}p^{k-1} + \cdots + j_1 p + j_0$ then it follows that $A^{(k)}_{i,j} = A^{(1)}_{i_0,j_0} A^{(1)}_{i_1,j_1} \cdots A^{(1)}_{i_{k-1},j_{k-1}}$ Similarly, for $B^{(k)}$ from observation after the description of its construction it follows that $B^{(k)}_{i,j} = B^{(1)}_{i_{k-1},j_0} B^{(1)}_{i_{k-2},j_1} \cdots B^{(1)}_{i_0,j_{k-1}}$ and since $A^{(1)} = B^{(1)}$ we can see that $\mathrm{a}^{(k)}_i = \mathrm{b}^{(k)}_{f(i)}$ where f is the permutation of $\{0, \ldots, p^k - 1\}$ which switches a number with its reversed in base p (using leading 0's where necessary). It is immediate from the above equations from $A^{(k)}_{i,j}$ and $B^{(k)}_{i,j}$ that $A^{(k)}_{i,j} = A^{(k)}_{f(i),f(j)}$ and $B^{(k)}_{i,j} = B^{(k)}_{f(i),f(j)}$ or, in matrix form $F^{(k)}A^{(k)}F^{(k)} = A^{(k)}$ and $F^{(k)}B^{(k)}F^{(k)} = B^{(k)}$, where $F^{(k)}$ is the permutation matrix corresponding to f. Since f is an order 2 permutation, $F^{(k)^{-1}} = F^{(k)^T} = F^{(k)}$. After this it is easy to prove the remaining:

b.) Using the above notation, we have to prove that $B^{(k)}B^{(k)^T}$ is a matrix that has all 0's under its secondary diagonal and nonzeros in the secondary diagonal itself. For $A^{(k)}A^{(k)^T}$ it is a slight generalization of what is in the proof of b.) for the k=1 case. On the other hand $A^{(k)} = B^{(k)}F^{(k)}$, so $A^{(k)^T} = F^{(k)}B^{(k)^T}$, so $A^{(k)}A^{(k)^T} = B^{(k)}F^{(k)}F^{(k)}B^{(k)^T} = B^{(k)}B^{(k)^T}$, so the above applies to $B^{(k)}B^{(k)^T}$, too.

c.) Again, the proof for $A^{(k)^2} = I$ is the same as for the $k = 1$ case and $B^{(k)^2} = A^{(k)}F^{(k)}A^{(k)}F^{(k)} = A^{(k)^2}$ since $F^{(k)}A^{(k)}F^{(k)} = A^{(k)}$

Uniqueness of subspaces:

We will show this by induction, too. The basic observation is that for each $\mathrm{b}_i(i = 0, \ldots, p^k - 2)$ there exists some $j = j^{(k)}(i)$ such that $T_j\mathrm{b}_i = \mathrm{b}_{i+1}$. For k=1 this is the case with $j^{(1)}(i) = 1$. For $k > 1$, $p \not| i + 1$, $j^{(k)}(i) = k$ from the construction, otherwise $j^{(k)}(i) = j^{(k-1)}(\lfloor i/p \rfloor)$. From this following the argument of the proof for the $k = 1$ case, the uniqueness of the chain of subspaces follows.

□

In [LRW2] there is another, shorter proof of this proposition that is based on a different numbering of the coordinates.

Application to the Level Stabilizer Factors

Let P be the canonical Sylow p-subgroup of $Sym(p^k)$. As we saw in Chapter III, we can map the factors of consecutive level stabilizers of P, $V_i = P_i/P_{i+1}$, to $GF(p)^{p^i}$, $i = 0, 1, \ldots, k-1$, via observing the amount of rotation on the size $k - i$ disks on the corresponding toy. In $Sym(p^k)$, because of the numbering of the positions on the toys, this means that the jth coordinate of the vector corresponding to hP_{i+1}, can be computed using the following formula:

$$v_{k,i}(h)[j] = \lfloor \frac{(jp^{k-i})^h - jp^{k-i}}{p^{k-i-1}} \rfloor \text{ for } j = 0, 1, \ldots, p^i - 1.$$

Here we identify the elements of $GF(p)$ with the integers modulo p. We prove the correctness of the formula by induction on k. For $k = 1$ the only possible value for both i and j is 0 and the formula obviously gives the amount of turn on the only disk. For $k > 1$ and $i = k - 1$, the denominator is 1 so we have to prove that the amount of rotation on the jth size 1 disk in the transformation corresponding to h is $v_{k,k-1}(h)[j] = (jp)^h - jp$. It is easy to see from the numbering process that the smallest number on that disk is jp. Since all bigger disks are fixed by h, $0 \leq v_{k,k-1}(h)[j] < p$, so this number is the amount of rotation of the disk. For $i < k-1$ we can think of the size 1 disks az dots on a D_{k-1} and the label of a dot corresponding to the size 1 disk B is $\lfloor m/p \rfloor$, where m is the D_k-label of any dot on B. Now $v_{k,i}(h)[j] = v_{k-1,i}(\overline{h})[j] = \lfloor \frac{(jp^{k-1-i})^{\overline{h}} - jp^{k-1-i}}{p^{(k-1)-i-1}} \rfloor = \lfloor \frac{\lfloor \frac{(jp^{k-i})^h}{p} \rfloor - jp^{k-1-i}}{p^{k-i-2}} \rfloor = \lfloor \frac{(jp^{k-i})^h - jp^{k-i})}{p^{k-i-1}} \rfloor$, where $\overline{h}$ denotes the action of h on the labels assigned to the size 1 disks by the above scheme, i.e. $m^{\overline{h}} = \lfloor \frac{(mp)^h}{p} \rfloor$.

The final observation that we want to make here is that there is a direct correspondence between the action of the canonical generators of P that do not fix V_i (i.e. $\tau_{k-i+1}, \ldots, \tau_k$) on the elements of V_i and the transformations $T_1, \ldots, T_i$, in the sense that T_j permutes the coordinates of $v_h \in V$ the same way as τ_{k-i+j} permutes the blocks corresponding to the size $k - i$ disks. Let $\sigma : P_i/P_{i+1} \rightarrow GF(p)^{p^i}$ be the homomorphism for which $\sigma(hP_{i+1}) = \mathrm{v}_h = \sum_{j=0}^{p^i-1} v_h[j]\mathrm{e}_j$. Then if $P_i \geq N \geq P_{i+1}$, $N \triangleleft P$ if and only if $\sigma(N)$ is invariant under the linear transformations $T_1, \ldots, T_i$. As we have shown that there is a unique series of such subspaces, we can conclude that there is a unique normal series for P refining the level stabilizer series into a chief series.

Let $P_i = N_0 > N_1 > \cdots > N_{p^i} = P_{i+1}$ the part of the chief series of P between two level stabilizers. Then for any $h \in P_i$ we can tell which is the smallest N_j that still contains h by expressing its image in $GF(p)^{p^i}$ as

$$\mathbf{v}_h = \sum_{l=1}^{p^i} c_l \mathbf{b}_l.$$

If j is the smallest index for which $c_j \neq 0$ then $h \in N_j$ but $h \notin N_{j+1}$. To compute the c_l's we have to multiply the coefficient-vector of $\mathbf{v}_h$ (expressed in the natural basis) by $B^{(i)^{-1}} = B^{(i)}$ to get the coefficient vector in the b-basis. There is, however, another step by step method for this purpose. Suppose we know already that $h \in N_j$ and we want to check whether $h \in N_{j+1}$. In this case, we can simply compute the dot product $a = \langle \mathbf{v}_h, \mathbf{b}_{p^i-j-1} \rangle$. Then $a = 0$ exactly when $h \in N_{j+1}$. Moreover, if we have $n_j \in N_j \setminus N_{j+1}$ such that $\langle \mathbf{v}_{n_j}, \mathbf{b}_{p^i-j-1} \rangle = 1$, then $hn_j^{-a} \in N_j$.

CHAPTER V

DATA STRUCTURE FOR p-GROUPS

In this chapter we will describe a data structure for permutation p-groups that corresponds to a chief series and provides a useful framework for algorithms that exploit that kind of series. The structure builds on the permutation action of the groups and uses the linear structure of the elementary abelian factors introduced in Chapters III and IV.

Definition 5.1

Let G be a group with a series of subgroups $G = G_0 > G_1 > \cdots > G_t = 1$. Let $S = \{g_1, g_2, \ldots, g_r\} \subset G$ be such that $\langle G_i \cap S \rangle = G_i$ for $i = 0, 1, \ldots t$, then we call S a strong generating set relative to the series. If $g_i \in G_j$ and $i < k \leq r$ imply $g_k \in G_j$, then we call $(g_1, g_2, \ldots, g_r)$ a strong generating sequence relative to $G = G_0 > G_1 > \cdots > G_t = 1$. □

For our data structure we will make use of a normal series of G that naturally arises from the permutation action of any permutation group. For intransitive groups, we show how to obtain a normal series using normal series for the transitive constituents.

Lemma 5.1

Let $G < Sym(\Omega)$ be a permutation group and let Δ_1 and Δ_2 be disjoint G-invariant subsets of Ω. Let $N \triangleleft G^{\Delta_2}$, and let $H < G$ be the subgroup that fixes all the points in Δ_1. Then $M = \{g \in H : g|_{\Delta_2} \in N\}$ is a normal subgroup of G.

Proof

$M = H \cap K$, where K is the kernel of the natural homomorphism from G to G^{Δ_2}/N. Since H is the kernel of the natural homomorphism from G to G^{Δ_1}, both K and H are normal and therefore so is their intersection. □

This observation gives us a way to compute normal series based on the orbit structure of the group G and normal series for the transitive constituents. Let $\Delta_1, \ldots, \Delta_t$ be the orbits of G and let G_i be the subgroup of G that fixes all points in the sets $\Delta_1, \ldots, \Delta_i$, then $G = G_0 \geq G_1 \geq \cdots \geq G_t = 1$, is a normal series for G and it can be further refined using Lemma V.1.

For transitive groups any block system provides a normal subgroup, namely the kernel of the action on the blocks. Thus, a hierarchy of finer and finer block systems provides a normal series. For the p-groups we will further refine this series.

The data structure that we will use for the p-group G, will consist of two series. One is a series of strong generators for the chief series $G = G_0 > G_1 > \cdots > G_t = 1$ that we get by refining the normal series defined by the orbits of G using the normal series obtained for the transitive constituents, utilizing the results of Chapters III and IV. The other part of the data structure is a series of homomorphisms from the level stabilizers of the transitive constituents into the vector spaces $GF(p)^{p^k}$. These maps will allow us to use vector space computations to decide whether a given $g \in P_i$ is in P_{i+1} and if not, which power z_i^m of the corresponding strong generator z_i should be factored out of it so that the residue, $z_i^{-m}g$, is in P_{i+1}.

In order to use the results of Section IV, we should embed each transitive constituent of G into the canonical Sylow p-subgroup $P^{(k)} \leq Sym(p^k)$. This means that we should assign labels to the points of the orbit Δ from the set $\{0, 1, \ldots, p^k - 1\}$ so that the action of G on the labels is a subgroup of $P^{(k)}$. We will do this recursively. Let $l(\omega)$ denote the label of the point $\omega \in \Omega$. For $k = 0$, the labeling is obvious: label the only point $\omega \in \Omega$ with 0. For $k \geq 1$, first compute a maximal block, Δ_1, and generators for $H = G^{\Delta_1}_{\{\Delta_1\}}$. Then label the points in Δ_1 recursively with the numbers $\{0, 1, \ldots, p^{k-1} - 1\}$ and fix $g_k \in G \setminus G_{\{\Delta_1\}}$. Label $\omega^{g_k^j}$ with $l(\omega) + jp^{k-1}$, for $\omega \in \Delta_1$ and $j = 1, 2, \ldots, p - 1$.

With this labeling, $G_{\{\Delta\}}$ is acting on the labels as a subgroup of $P^{(k)}_{\{\{0,\ldots,p^{k-1}\}\}}$, the subgroup of $P^{(k)}$ that setwise fixes the block $\{0, 1 \ldots, p^{k-1} - 1\}$ (and therefore all blocks $\{jp^{k-1}, jp^{k-1} + 1, \ldots, (j + 1)p^{k-1} - 1\}$ for $j = 0, 1, \ldots, p - 1$). We show this by induction on k. For $k = 0$ it is trivial. Let us suppose that for $j < k$ this labeling gives the desired result. We want to show that if $|\Delta| = p^k$, $G_{\{\Delta\}}$, labeled by the above algorithm, acts on the labels as a subgroup of the canonical p-subgroup of $Sym(p^k)$. The action on Δ_1 has this property because of the inductive hypothesis. The actions on the other blocks can be obtained by conjugation by g_k^i and τ_k^i, respectively, and both of these transform the labels of Δ_1 the same way. It remains to be shown that g_k's action on the labels is in $P^{(k)}$. To see this, observe that $\tau_k^{-1} g_k$ fixes all labels $\geq p^{k-1}$, while for the labels in $\{0, \ldots, p^{k-1} - 1\}$ the action of it is the same as that of g_k^p. Since g_k^p fixes Δ_1, $g_k^p \mid \Delta_1$ acts on $\{0, \ldots, p^{k-1}\}$ as an element of $P^{(k-1)}$. Therefore $\tau_k^{-1} g_k \in P^{(k)}$, which means that g_k's action on the labels is indeed in $P^{(k)}$, too.

Computing the Series of Maps

Let $G = G_0 \geq G_1 \geq \cdots \geq G_r = 1$ be a chief series for G obtained by the procedure described above, that is, this series is a refinement of the orbit stabilizer series using the embedding of the transitive constituents into the canonical Sylow p-subgroups for obtaining normal series for the transitive constituents. Then each G_i fixes all the points in the orbits $\Delta_1, \Delta_2, \ldots, \Delta_{j(i)-1}$ but not $\Delta_{j(i)}$, $|\Delta_{j(i)}| = p^{k(i)}$, the group $G^{\Delta_{j(i)}}$ is embedded into the canonical Sylow p-subgroup $P^{(k(i))}$ of $Sym(p^{k(i)})$ and for the image H_i of $G_i^{\Delta_{j(i)}}$, $P_a \geq H_i > P_b$, where P_a and P_b are successive level stabilizers in $P^{(k(i))}$, with $P_a/P_b \cong V_i = GF(p)^{d(i)}$, where $j(i)$, $k(i)$ and $d(i)$ are defined by the above.

The homomorphisms that map the elements of each G_i into the corresponding vector space $V(i)$ can be computed as maps $\sigma_i : G_i \to V(i)$. The computation of $\sigma(g)$ is done in two steps. First, map the points of $\Delta_{j(i)}$ to the labels $\{0, 1, \ldots, p^{k(i)} - 1\}$ as shown above, call this map $\theta : \Delta_{j(i)} \to \{0, 1, \ldots, p^{k(i)} - 1\}$. Then compute the coordinates corresponding to $\theta(g)$, using the formula from Chapter IV. The first part of the computation is meaningful for all $g \in G$, but the second part makes sense only if g is in the level stabilizer corresponding to G_i.

For the first part, θ can be represented as two arrays. One holds the values $\theta(\omega_m)$ for $\omega_m \in \Delta_{j(i)}$, the other for the inverse images $\theta^{-1}(m)$ for $m = 0, 1, \ldots, p^{k(i)} - 1$. Then the action of $g \in G$ on these labels can be computed by using two table look-ups per point in addition to the computation of the action of g (which is another table look-up if we use the usual representation of permutations).

For the second part, we use a constant number of arithmetic operations per coordinate.

Note that when we want to compute the vector representation of some gh^m, $g, h \in G_i$, $0 \leq m < p$, where we know vector representations for g and h, we do not have to start from scratch, since σ_i is a homomorphism, so $v_{gh^m} = v_g + mv_h$. This observation is very useful to speed up computations while still working in the same vector space. We need to recompute vectors only when crossing the level stabilizer boundaries.

Computing the Series of Strong Generators

Let P be the direct product of the canonical Sylow p-groups corresponding to the orbits. Then for P we have the chief series without much computation, because we can compute permutations corresponding to vectors by observing the movement of the labels on the toy, turning the disks according to the coordinates of the vector, and we have to use the row vectors of the appropriate $B^{(i)}$ matrices, to get inverse images of the P-invariant subspaces. Let us denote the chief series of P by $P = P_0 > P_1 > \cdots, P_t = 1$, then $|P_i/P_{i+1}| = p$, so for any $p_{i+1} \in P_i \setminus P_{i+1}$, $P_i = \langle p_{i+1}, P_{i+1} \rangle$, therefore, for generating the P_i's, it is enough to have the inverse image of the generators of the invariant subspaces.

If G is embedded into P by the injection $\sigma : G \to P$ then for $H = \sigma(G)$ we can get a chief series for H by considering the distinct subgroups in the series $H_0 = H \cap P_0 \geq H_1 = H \cap P_1 \geq \cdots \geq H_t = H \cap P_t = 1$. Let $h_{i+1} \in H_i \setminus H_{i+1}$ if $H_i > H_{i+1}$, then $h_{i+1} \in P_i \setminus P_{i+1}$, too, so we can replace p_{i+1} by h_{i+1} in the strong generating sequence of P, and we still have a strong generating sequence. This shows that either $H_i P_{i+1} = P_i$, in which case we say that H covers the factor P_i/P_{i+1}, or $(H \cap P_i) \setminus P_{i+1} = \emptyset$ in which case we say that H avoids that factor.

If we have a sequence $h_1, h_2, \ldots, h_t$, such that $h_{i+1} = 1$ if H avoids P_i/P_{i+1} and

$h_{i+1} \in H_i \setminus H_{i+1}$ if H covers P_i/P_{i+1}, then we can use this sequence to sift through H, i.e. to determine whether $g \in P$ is in H or not. Figure 2 shows the algorithm. If the algorithm returns **true** then $g = h_1^{m_1} h_2^{m_2} \cdots h_t^{m_t}$ where for the m_i's not defined

```
{input: A strong generating sequence for H,
        (h_1, ..., h_t), and g ∈ P}
{output: true if g ∈ H, false otherwise}
begin
i := 1
while $i < t$ and $g \neq 1$ do
    if $h_i = 1$ and $g \notin P_i$ then return false
    Compute $m_i$ such that $h_i^{-m_i} g \in P_i$
    $g := h_i^{-m_i} g$
return true
end.
```

Figure 2: Sifting using a strong generating sequence

by the algorithm we can use any integer, since then $h_i = 1$. This shows that if the algorithm returns **true** for g then $g \in H$. On the other hand, if the algorithm returns **false** then there is $h \in H$ such that $gh \in P_{i-1} \setminus P_i$ and H avoids the factor P_{i-1}/P_i, which means $gh \notin H$, that is, $g \notin H$. We can compute m_i by computing the vector representation of g and taking the leading coefficient of it, provided that the vector representation of h_i has leading coefficient 1.

Now imagine that we have a series $h_1, h_2, \ldots, h_t$ such that $h_i \in (H \cap P_{i-1} \setminus P_i) \cup \{1\}$ for $i = 1, 2, \ldots, t$, and we want to show that this is a strong generating sequence for H, that is, the distinct elements of the series $H_0 \geq H_1 \geq \cdots \geq H_t$, where $H_i = \langle h_{i+1}, \ldots h_t \rangle$, constitute a chief series for H. For this, we have to show that $H_i \triangleleft H$ and that $h_i^p \in H_i$ for $i = 1, 2, \ldots t$. The first condition means that we have a normal series, the second, that for the nontrivial steps, i.e. when $H_{i-1} \neq H_i$, $|H_{i-1} : H_i| = p$. To verify that these conditions are satisfied, we have to sift h_i^z, for each generator z of H (using any generator set for H) and for each $i = 1, \ldots, t$ through the series for H_{i-1}, and sift h_i^p for each $i = 1, \ldots, t$ through the series for H_i. If everything sifts through, we have a strong generating sequence.

We can use this sifting proof with a slight modification to obtain the strong generating sequence from a set of generators for H. Start with a sequence of all 1's and a stack containing the generators of H. Modify the sifting procedure of Figure 2 so that instead of returning **true**, it returns $t + 1$ and instead of returning **false**, it inserts the current value of g into the sequence and returns the position of the failure. Let us call this event "g got stuck at level i". In this case, push h_i^p to the stack along with h_i^z, for all generators z of H, noting that the sifting should start at level i and $i - 1$, respectively, this is because we know that the stuck element is in P_{i-1} which is normalized by H. Repeat the process with the first element on the stack, until the stack gets empty. Figure 3 shows the algorithm. When this algorithm finishes, we have sifted all the elements necessary to check for proving that the "stuck" elements constitute a strong generating system for H. If we have had started with what the procedure returned, everything would have sifted through, since in this case the stuck elements would have been multiplied by their own inverse in the process.

Complexity

The number of elements that get stuck is exactly the length of the chief series for H, that is $O(n)$, if n denotes the size of the permutation domain. Therefore we sift $O(ns)$ elements, where $s = |S|$. Each sift costs $O(n^2)$ work, because it takes $O(n)$ work to get from one level to the next and there are $O(n)$ levels. To compute

```
{input:   $H \leq P$, given by the generating set $S$
          $P$ has chief series $P = P_0 > P_1 > \cdots > P_t = 1$ }
{output: a strong generating system for $H$}
begin
$T :=$ emptystack()
$(h_1, h_2, \ldots, h_t) = (1, 1, \ldots, 1)$
for all $z \in S$ do $push((z, 1), T)$
while $T$ is not empty do
        $(g, j) := pop(T)$
        $i := sift(g, j, (h_1, h_2, \ldots, h_t))$
        if $i \neq t + 1$ then
           { $sift()$ already inserted the new strong generator $h_i$ }
           $push((h_i^p, i + 1), T)$
           for all $z \in S$ do
                    $push((h_i^z, i), T)$
return $(h_1, h_2, \ldots, h_t)$
end.
```

Figure 3: Computing the strong generating sequence

the maps to set up the structure takes $O(sn^2)$ time, including finding orbits, blocks, and the embeddings of the transitive constituents into the canonical Sylow p-groups. Therefore the whole complexity of the computation of the strong generating sequence is $O(sn^3)$.

CHAPTER VI

RECOGNIZING NILPOTENCE OF PERMUTATION GROUPS

In the previous chapters we developed a data structure that can be used for p-groups, and we will see that it can be used for fast algorithms dealing with nilpotent groups. In order to use these algorithms, however, we have to be sure that the groups to which we want to apply them, are in fact nilpotent. In this chapter we will see a very fast way to answer this question by basically just looking at the generators of the group, computing only orbit and block structures. The tests are fast by both theoretical and practical measures, much faster than computing the size of the group. Thus we can use these tests to decide whether we are able to use the faster nilpotent group algorithms, or we should fall back to a general algorithm. As nilpotent permutation groups are direct products of their Sylow p-subgroups, testing a group for nilpotency can be done by using this property.

The idea of using the orbit/block structure of groups for reducing the testing problem to the primitive case is not new, e.g. McKenzie uses it to prove that the problem of testing whether a group is a p-group is in NC [Mc]. Here we provide details of a test that runs faster than the algorithm for finding a block system (for the transitive case).

The material presented in this chapter has been published in [Rá] .

The p-Group Test

The Algorithm

Let p be a prime. In this section, we characterize transitive p-groups and describe an algorithm for their recognition.

Lemma 6.1

Let G be a transitive permutation group. Then G is a p-group iff

1. G has a p-element block system $\Delta = \{\Delta_1, \Delta_2, \ldots, \Delta_p\}$
2. G acts on Δ cyclically
3. If G_1 is the subgroup of G that stabilizes the blocks in Δ then $G_1^{\Delta_1}$ is a p-group

Proof

We have already seen in Chapter III that a transitive p-group has the stated properties.

To show that (1)-(3) implies that G is a p-group, it is enough to show that (3) implies that not only the action of G_1 on Δ_1, but the subgroup itself is a p-group. This follows from the fact that the action of G_1 on each of the other blocks is isomorphic with its action on Δ_1 (by conjugation via an element of G that maps Δ_1 to the given block), so G_1, being a subgroup of the direct product of p-groups, is a p-group. □

Combining this with the added observation that if G is transitive, then the action of G_1 on Δ_1 is also transitive (because if an element of G takes $\omega \in \Delta_1$ to

$\tau \in \Delta_1$, that element should be in G_1, since from (2) it follows that if an element of G stabilizes one block, it stabilizes all of them), we get that the algorithm shown on Figure 4 tests whether (the transitive) G is a p-group. Some details are explained in the following subsections.

```
IsTransitivepGroup($S$)
{input:  $G$, a transitive permutation group on
        $\{\omega_1, \ldots, \omega_{p^k}\}$, given by a set of generators $S$}
{output: true if $G$ is a $p$-group, false otherwise }

begin
if $G$ is the trivial group (i.e. k=0) then return true
if $k = 1$ then
  if IsCyclic($G$) then return true
  else return false
$\Delta$:=MaximalBlocks($S$)
if Size($\Delta$) $\neq p$ then return false
Let $g \in S$ be the first generator that does not
  stabilize $\Delta_1$
if $g$ does not permute the blocks cyclically then
  return false
for each generator $g_i \in S$ do
  Find $l$ such that $g^l g_i$ stabilizes $\Delta_1$
  if $g^l g_i$ does not stabilize the other blocks then
    return false
Let $S$ be the set of (Schreier) generators for the $\Delta_1$-
  constituent of the subgroup of $G$ that stabilizes the
  blocks
return IsTransitivepGroup($S$)
end.
```

Figure 4: Algorithm for testing whether a group is a p-group

Finding Maximal Blocks

The algorithm on Figure 4 calls the procedure **MaximalBlocks**, which is supposed to return a block system consisting of maximal blocks, if the group generated by S was a p-group, and any system of blocks otherwise. The best result known for finding a nontrivial block system for an arbitrary group is $\Omega(|S|n^2)$, and there is another algorithm of Beals that runs in time $O(n \log^3 |G|)$, which is $O(n \log^c n)$ for "small base" groups, that is for groups that have a base of size less than $\log^{c'} n$ [Be]. Schönert and Seress [SchSe] report a similar result, with implementation of the algorithm in GAP. They state that they find a minimal block in time $O(n \log^3 |G| + ns \log |G|)$, s being number of generators.

Unfortunately, p-groups do not fall in the small base group category, as the base for a p-group on p^k points can be as big as p^{k-1}.

Atkinson's test of primitivity [At] involves an algorithm (we will call it **Blocks**(S, Δ)) for finding a block system in which one of the blocks is the smallest block that contains the set Δ (see [Bu] for practical implementational remarks). Therefore, if we know that there exists a nontrivial block that contains the points of Δ ($|\Delta| > 1$) then **Blocks**(S, Δ) will return a system of nontrivial blocks. The running time of **Blocks** is $O(\alpha(n, 4|S|n))$. Here $\alpha(x, y)$ denotes the time required for x Union and y Find operations in a Union-Find data structure, the asymptotically best implementation of which runs in time $O(x \log^*(x+y))$ (see [Tar], he proves an asymptotically somewhat stronger result in terms of the inverse Ackermann function, we use $\log^*$ here for easier formulation of the final time bound). An implementation of the algorithm can be found in GAP [Sch] (namely **Blocks**$(< G >, < D >, < \mathit{seed} >)$).

The following proposition provides foundation for finding a suitable set Δ:

Lemma 6.2

Let G be a p-group acting transitively on the set Ω. Let $g, h \in G$. Then the blocks in $\mathbf{\Delta}$ of Lemma 6.1 are stabilized by $[g, h]$.

Proof

The action of G on $\mathbf{\Delta}$ is cyclic, so the action of the commutator is trivial. That means that each orbit of $[g, h]$ is entirely contained in one of the Δ_i's. □

So, taking the points of any nontrivial orbit of the commutator of any two elements of a transitive p-group will provide us with the seed for Atkinson's algorithm. If we find two elements that do not commute then the commutator will have a nontrivial orbit. However, we do not have to find two noncommuting elements to find blocks. If we have an element z of the centre that does not generate a transitive subgroup, the orbits of $\langle z \rangle$ constitute a block system, since $\langle z \rangle$ is normal. Finally, if we find a central element z that does generate a transitive subgroup of size greater than p then the z^p will generate a nontrivial nontransitive normal subgroup.

Figure 5 shows the algorithm for finding a system of maximal blocks for p-groups. It will return a system of blocks in the non-p-group case, too, but that system might not consist of maximal nontrivial blocks. Note that **MaximalBlocks** is never called with G being a p-element group, except for the recursive call in it.

Timing

In timing arguments, n will always mean the size of the permutation domain, and we will use $s = |S|$. First, we give the timing for **MaximalBlocks**. Finding whether there is an element in the generating set that does not commute with the first

```
MaximalBlocks($S$)
{input:  $G$, a transitive permutation group on
    $\{\omega_1, \ldots, \omega_{p^k}\}$, given by a set of generators $S$}
{output:  A nontrivial system of maximal blocks, if
    $G$ is a $p$-group, a (possibly trivial) system of (not
    necessarily nontrivial) blocks otherwise }

begin
if $k = 1$ then return $\{\{\omega_1\}, \{\omega_2\}, \ldots, \{\omega_p\}\}$
Let $g$ be the first element of S.
if $\exists h \in S$ such that $[g, h] \neq 1$ then
  $g := [g, h]$
  Let$\Delta$ be a set consisting of the points in a nontrivial
      orbit of $g$
  $\Delta :=$ Blocks($S, \Delta$)
else
  if $ord(g) = p^k$ then $g := g^p$
  $\Delta :=$ Orbits($\langle g \rangle$)
if $|\Delta| \leq p$ then return $\Delta$
Let $T$ be a generating set for the action of $G$ on the
  blocks in $\Delta$
$\Gamma :=$ MaximalBlocks($T$)
Unify the blocks in $\Delta$ that are in the same block in $\Gamma$
return the resulting block system
end.
```

Figure 5: Algorithm for computing maximal blocks for a p-group

generator costs $O(s)$ permutation multiplications and inversions, so it takes $O(sn)$ time. If we find such an element, the call to **Blocks** takes $O(\alpha(n, 4sn))$ time, in the other case the call to **Orbits** takes even less, $O(sn)$. Computing a generating set for the action takes $O(sn/p)$ time, and the unification of the blocks at the end takes $O(n)$. The recursive call is with input size at most $1/p$ of the original input size, so the whole procedure takes $O(\alpha(n, 4sn))$ time.

For **IsTransitivepGroup**, in the nonrecursive case it takes $O(sn)$ (in this case $n = p$) time to find out whether S generates a cyclic group. As we saw, **MaximalBlocks** takes $O(\alpha(n, 4sn))$. To check whether a generator permutes the blocks cyclically, costs $O(p)$, and for each generator in the **for** loop finding l takes constant time since we could mark the blocks with the power of g that moves the first block there while checking whether g permutes the blocks cyclically. Then finding out whether $g^l g_i$ stabilizes all of the blocks takes $O(p)$ time again, giving $O(sp)$ cost for the **for** loop. Finding the generators for the Δ_1-constituent of the subgroup stabilizing the blocks is tricky, but it can be done in time $O(sn)$. These generators are the Schreier generators for G_1 (that are of the form $g^k s(g^{-1})^m$, $0 \leq k < p$ and m such that the product setwise fixes the blocks), restricted to Δ_1. First, we compute where g^k moves the points of Δ_1 for $0 \leq k < p$, and at the same time we mark δ^{g^k} with δ, this way we can tell with a couple of table lookups $\delta^{g^k s(g^{-1})^m}$ with the appropriate m, for each $\delta \in \Delta$. After this preprocessing (which costs $O(n)$) we can compute the Δ_1-constituent of the Schreier generators in the time necessary to write them down, i.e. $O(sn)$, since there are at most ps of them, each being of size n/p. Since we have a recursive call, this gives us the following recursive formula for the timing: $T(s, p^k) = T(ps, p^{k-1}) + O(\alpha(p^k, 4sp^k))$. Since $\alpha(p^k, 4sp^k) \geq \alpha(p^{k-1}, 4psp^{k-1})$, and the right hand side term will appear when we express $T(ps, p^{k-1})$, we get $T(s,n) = O(\log n\alpha(n, 4sn))$.

The Nilpotency Test

The Algorithm

As in the p-group case, first we characterize transitive nilpotent groups in a way that directly leads to our recognition algorithm. A nilpotent group is the direct product of its Sylow-p-subgroups, so in particular, let G be a nilpotent group, then $G = P \times P'$, where P is a p-group and P' is a nilpotent p'-group (a p'-group is a group the order of which is not divisible by p). Furthermore, if $G = \langle S \rangle$, and for $g \in G$ we define g_p and $g_{p'}$ the following way: let us denote the order of g by $o(g)$; if $o(g) = p^k r$, where p does not divide r, then let $g_p = g^r$ and let $g_{p'} = g^{p^k}$, then $P = \langle \{s_p : s \in S\} \rangle$ and $P' = \langle \{s_{p'} : s \in S\} \rangle$. Trivially, both P and P' are normal in G.

If, in addition, G is transitive, then the orbits of both P and P' are blocks for G. The lengths of the orbits of P are p-powers, and the lengths of the orbits of P' are not divisible by p.

Figure 6 shows the algorithm for testing nilpotence.

Correctness of the Algorithm

Lemma 6.3

The function **IsTransitiveNilpotent** returns **true** for all nilpotent groups and **false** for all others.

Proof

By the characterization given above, **IsTransitiveNilpotent** returns **true** for all nilpotent groups. The only thing to show is that if the function

```
IsTransitiveNilpotent(S)
{input:  $G$, a transitive permutation group on
      $\{\omega_1, \ldots, \omega_n\}$, given by a set of generators $S$}
{output: true if $G$ is nilpotent, false otherwise }

begin
if $n = 1$ then return true
Find a prime $p$ that divides $n$
if $n = p^k$ for some k then
  return IsTransitivepGroup($G$)
Compute the $p$ and $p'$ parts of the generators,
  $P$ and $P'$
if the orbits of $\langle P \rangle$ do not form a block system for $G$
  then return false
if the orbits of $\langle P' \rangle$ do not form a block system for $G$
  then return false
if the length of an orbit of $\langle P \rangle$ is not a $p$-power
  then return false
if $p$ divides the length of an orbit of $\langle P' \rangle$
  then return false
Let $T$ be a set of generators for the action of $G$ on
  the orbits of $\langle P \rangle$
Let $U$ be a set of generators for the action of $G$ on
  the orbits of $\langle P' \rangle$
return IsTransitivepGroup($T$) and
        IsTransitiveNilpotent($U$)
end.
```

Figure 6: Algorithm for testing nilpotence

returns **true**, G was in fact a nilpotent group. To show this, it is enough to prove that if the function does not return **false**, then G is the direct product of P and P'. Since the orbits of P (P') are blocks for G, P (P') is contained in the normal subgroup that stabilizes those blocks. Let us denote this normal subgroup by N (N'). Now $\langle N, N' \rangle = G$, since $\langle P, P' \rangle = G$. On the other hand, $M = N \cap N'$ is normal in N (N'). So for each orbit of N (N'), the restriction of M to that orbit is normal in the restriction of N (N') to the same orbit, hence the orbits of the restriction of M are blocks for the restriction of N (N'), so the size of them divides the size of the orbits of N (N'). Finally, since the sizes of orbits of N and N' are relatively prime to each other, we can conclude that M's orbits are of length 1 which means that M is the trivial group, so $G = N \times N'$. But $P \leq N$, $P' \leq N'$, $\langle P, P' \rangle = G$ $N \cap N' = 1$ implies that $P = N$, $P' = N'$. $\square$

Timing

Factorization of n is $O(\sqrt{n}\log^2 n)$ (even by the brute force method). The p-group test is in $O(\log n \ \alpha(n, 4sn))$ as shown above. Factoring the generators seems to require to compute the order of them, which can be a large number in the general case, but for a transitive nilpotent group it is a divisor of n. (For p-groups it follows from the fact that each cycle of an element of a permutation p-group has p-power length and $n = p^k$, and if we have a nilpotent group that is not a p-group then we can factor it to a p-group acting on p^k points and a nilpotent group acting on r points, where $n = p^k r$.) So we first check whether $g^n = 1$ for each generator g. This can be done in $O(n \log n)$ time per generator. If any of them fails the test, the group is not nilpotent, otherwise, we can do the factoring in $O(n \log n)$ time. Computing the orbits is $O(sn)$. Checking whether a given partition is a block system uses $O(sn)$ time. Computing actions (given the block system) costs $O(sm)$ where m is the number of blocks. The call to **IsTransitivepGroup** takes time $O(k\alpha(p^k, 4sp^k))$, and finally, the recursive call is made on an input sized at most $1/p$ times the original input size, so it only affects the constant in the timing of the whole algorithm. Thus, we can conclude that the complexity of the whole algorithm is $O(\log n \ \alpha(n, 4sn))$, just as it was for the p-group case.

Implementation

We tested our algorithm by programming it in GAP [Sch], running on a Sun Sparc10 computer, under the operating system SunOS 4.1.3. Since there is no function in GAP that tests whether a group is p-group or not (although it is easy to write – one can just factors the size), we compared the nilpotence testing functions. GAP's IsNilpotent function computes the lower central series of the group and returns **true** iff the last element of that series is the trivial group.

The tests were conducted for both nilpotent and non-nilpotent subgroups of $Sym(100)$. The nilpotent groups were subgroups generated by 2 or 3 elements of direct products of a p_1-group and a p_2-group, acting on disjoint sets, having the direct product act on the union of the two domains. Other nilpotent groups were cyclic groups and p-groups. We measured GAP's time after the computation of the point stabilizer series. We found that for the cyclic groups, GAP gives a result sooner, but in all the other cases we found our program working faster, our test never took longer than 2 seconds, while, for example, it took more than 5 minutes for GAP to compute the lower central series for a Sylow-2-subgroup of $Sym(80)$. For smaller groups the times were closer.

For non-nilpotent groups, we used wreath products of p-groups with the 2- or 3-element group and the primitive groups from GAP's group library (which contains primitive groups on up to 50 points). For these groups, our algorithm showed to perform even more favorably: for similar size groups it gives a rejecting result even faster than an accepting one, while it seems that for GAP it is harder to reject than to accept.

CHAPTER VII

SOME ALGORITHMS FOR NILPOTENT GROUPS

In this chapter we will see some examples where the data structure that we presented in Chapter V is a natural one to use. We can decompose nilpotent groups into the direct product of their Sylow p-subgroups, as we saw in Chapter VI. For the problems in this chapter, the answer is always the direct product of the answers involving the p-components, so we will present the algorithms for p-groups.

The material presented in this chapter has been published in [LRW2] .

The Normalizer Algorithm

The Problem

Given $K \leq Sym(\Omega)$, K nilpotent, and $H \leq K$, $G \leq K$. Find $N_G(H) = \{g \in G : H^g = H\}$, the normalizer of H in G.

If $K = K_1 \times K_2$ then $H = H_1 \times H_2$ and $G = G_1 \times G_2$, where $H_1, G_1 \subset K_1$ and $H_2, G_2 \subset K_2$ and for $(h_1, h_2) \in H_1 \times H_2$ and $(g_1, g_2) \in G_1 \times G_2$ we have $h^g = (h_1^{g_1}, h_2^{g_2})$, and $N_{G_1 \times G_2}(H_1 \times H_2) = N_{G_1}(H_1) \times N_{G_2}(H_2)$, so we can indeed decompose this problem to the p-components.

The Overall Design of the Algorithm

From here on we will describe how to solve the normalizer problem for p-groups. Let therefore K be a p-group for some prime p and $H, G \leq K$. Let furthermore K

have the chief series $K = K_0 > K_1 > \cdots > K_t = 1$, that is $K_i \triangleleft K$ and $|K_{i-1}/K_i| = p$ for $i = 1, \ldots, t$. Then this is necessarily a central series, i.e. for $k_{i+1} \in K_i$ and $k \in K$, $[k_{i+1}, k] \in K_{i+1}$. This is a consequence of the fact that the action of K by conjugation on the factors K_i/K_{i+1} is necessarily trivial, since it necessarily fixes one of the p cosets (the one that contains the identity) and then there is no room for an orbit of length p on the remaining $p-1$ cosets.

We define a series $H_0 \geq H_1 \geq \cdots \geq H_t$ for H by $H_i = H \cap K_i$ for $i = 0, \ldots, t$. Now the plan of the algorithm is to compute successively $N_G(H_i)$, starting from the end, i.e. with $N_G(H_t) = N_G(1) = G$ and working towards $N_G(H_0) = N_G(H)$. To go from $N_G(H_{i+1})$ to $N_G(H_i)$, if $H_i \neq H_{i+1}$, we will compute in succession $N_G(H_iK_j) \cap N_G(H_{i+1})$, in this case starting from $j = i$, $N_G(H_iK_i) \cap N_G(H_{i+1}) = N_G(K_i) \cap N_G(H_{i+1}) = G \cap N_G(H_{i+1}) = N_G(H_{i+1})$, and finally getting $N_G(H_iK_t) \cap N_G(H_{i+1}) = N_G(H_i) \cap N_G(H_{i+1}) = N_G(H_i)$. Let $M = N_G(H_iK_j) \cap N_G(H_{i+1})$, then $N_M(H_iK_{j+1}) = N_G(H_iK_{j+1}) \cap N_G(H_{i+1})$, since each $g \in G$ that normalizes H_iK_{j+1} normalizes H_iK_j, too, so this is true in particular for the elements of $N_G(H_{i+1})$. With the observations that H_{t-1} is necessarily central in K, so $N_G(H_{t-1}) = G$, and that $[H_i, G] \leq [K_i, K] = K_{i+1}$, so $N_G(H_iK_i) = N_G(H_iK_{i+1})$, we proved that the algorithm shown in Figure 7 computes $N_G(H)$.

Updating the Normalizer

The heart of the algorithm is the computation of $N_M(H_iK_{j+1})$. We will show that this normalizer is either M itself, or it is a maximal subgroup of it. What we know at this point of the algorithm, either as result of the initial conditions, or as consequence of previous computations, is the following:

```
{input:  Subgroups $G$ and $H$ of a finite $p$-group $K$.
         A chief series $K = K_0 \geq \cdots \geq K_t = 1$ of $K$.
         $H_i = K_i \cap H$ for $i = 0, \ldots, t$.}
{output: $N_G(H)$.}
begin
$M := G$
for $i := t - 2$ downto $0$ and $H_i \neq H_{i+1}$do
    for $j := i + 1$ to $t - 1$ do
        $M := N_M(H_i K_{j+1})$
return $M$
end.
```

Figure 7: The normalizer algorithm for p-groups.

a.) M normalizes H_{i+1}

b.) M normalizes $H_i K_j$

c.) $0 \leq i < j < t$

d.) $HK_i = HK_{i+1}$

Now, if $HK_j = HK_{j+1}$, then $H_i K_j = H_i K_{j+1}$, so M already normalizes $H_i K_{j+1}$, so we have to deal only with the case $H_j = H_{j+1}$. In this case $V = H_i K_j / H_{i+1} K_{j+1}$ is of order p^2, and since it has two different nontrivial subgroups ($H_i K_{j+1} / H_{i+1} K_{j+1}$ and $H_{i+1} K_j / H_{i+1} K_{j+1}$), it is elementary abelian, so it can be thought of as a vector space of dimension 2 over $GF(p)$, with basis $\{h_{i+1} H_{i+1} K_{j+1}, k_{j+1} H_{i+1} K_{j+1}\}$, where $h_{i+1} \in H_i \setminus H_{i+1}$ and $k_{j+1} \in K_j \setminus K_{j+1}$. Since M normalizes H_{i+1}, M acts on this vector space as linear transformations. The matrix of the transformation corresponding to $m \in M$ is an upper triangular matrix with 1's on the diagonal, because $h_{i+1}^m = h_{i+1}[h_{i+1}, m]$, where $[h_{i+1}, m] \in H_{i+1} \subset H_{i+1} K_j$, and $k_{j+1}^m = k_{j+1}[k_{j+1}, m]$, where $[k_{j+1}, m] \in K_{j+1} \subset H_{i+1}K_{j+1}$ This also means that the map $\theta : M \to GF(p)$, for which $k_{i+1}^{\theta(m)} = [h_{i+1}, m] \bmod H_{i+1}K_{j+1}$, is a homomorphism. The elements of $N_M(H_iK_{j+1})$ are exactly those m's for which $\theta(m) = 0$, therefore, indeed, they constitute either a maximal subgroup of M, or the whole M.

If we have a representation of M that consists of a strong generating system for M corresponding to a chain of subgroups, $M = M_0 > M_1 > \cdots > M_r = 1$, where $|M_{i-1} : M_i| = p$, then we can find a similar system for $\overline{M}$, a maximal subgroup of M, using the following algorithm. Let $m_1, \ldots, m_r$ be a strong generating system for the above series, $m_i \in M_{i-1} \setminus M_i$. Find $s = \min\{i \in \{1, \ldots, r\} : m_j \in \overline{M}$ for $i \le j \le r\}$. For $i = 1, \ldots, s-1$, find α_i such that $\overline{m_i} = m_i m_s^{\alpha_i} \in \overline{M}$. Then $\overline{m_1}, \ldots, \overline{m_{s-1}}, m_{s+1}, \ldots, m_r$ will be a strong generating sequence for $\overline{M}$. Such α_i's exist, since $\overline{M}$ is maximal in M and therefore the right multiplication action of M on the cosets of $\overline{M}$ is cyclic. To prove that the above sequence is a strong generating sequence corresponding to the series $M_0 \cap \overline{M}, \ldots, M_{s-1} \cap \overline{M}, M_{s+1} \cap \overline{M} = M_{s+1}, \ldots, M_r \cap \overline{M} = M_r = 1$, it is enough to show that $|M_i : \overline{M_i}| = p$ for $i = 1, \ldots, s-1$, where $\overline{M_i} = \langle \overline{m_i}, \ldots, \overline{m_{s-1}}, m_{s+1}, \ldots, m_r \rangle$. This follows from $\langle m_s, \overline{M_i} \rangle = M_i$ and $m_s^p \in M_s \subset \overline{M_i}$.

To implement the above algorithm, we have to

a.) compute a strong generating system $m_1, \ldots, m_r$ for M.

b.) compute $\theta(m)$ for $m_i \in M$.

c.) find s for which $m_s \notin N_M(H_iK_{j+1})$.

d.) compute α_i's such that $m_i m_s^{\alpha_i} \in N_M(H_iK_{j+1})$.

Since during the course of the algorithm a strong generating sequence for M is maintained, it is enough that we compute such a sequence for the initial step, i.e., when $M = G$. The data structure proposed in Chapter VI is even more than what we need (we do not require the corresponding subgroup-chain to be a normal series).

We compute θ as follows. We need to express $[h_{i+1}, m]$ as $k_{j+1}^{\phi} w$ where $w \in H_{i+1}K_{j+1}$. As k_{j+1} commutes with K modulo K_{j+1}, this is equivalent to finding some $\overline{h} \in H_{i+1}$ such that $[h_{i+1}, m]\overline{h}^{-1} \in K_i$, and then find the leading exponent, ϕ, of its Φ_i-image in $GF(p)_i^d$ (using the notations of Chapter VI). With this notation, $\theta(m) = \phi$. Of course, for H we use the same data structure. We could compute $\overline{h}$ by sifting $[h_{i+1}, m]$ through the H-structure, but we can do better, as the following argument shows. We will keep an additional permutation x with each m, for which $[h_{i+1}, m] \equiv x \pmod{K_j}$, and we will update it so that this congruence will be true modulo K_{j+1} after the updating step. If $HK_j = HK_{j+1}$, as we saw, the normalizer does not change, but we have to update x, using a sifting step. In the other case, if there is no m in the strong generating sequence of M for which $\theta(m) \neq 0$ then M normalizes H_iK_{j+1} and no computation has to be done. Otherwise, let $\overline{m}$ be the element with the largest index in that sequence for which $\theta(\overline{m}) \neq 0$. For each preceeding m we first compute the α for which $\theta(m\overline{m}^{\,\alpha}) = 0$. As θ is a homomorphism, $\theta(m\overline{m}^{\,\alpha}) = \theta(m) + \alpha\theta(\overline{m})$, which we can solve for α, since $\theta(\overline{m}) \neq 0$. So we update m, $m_{new} = m\overline{m}^{\,\alpha}$. After this we update x: $x_{new} = (\overline{x}\,\overline{m}^{\,-1})^{\alpha} x\overline{m}^{\,\alpha}$. We have to show that $x_{new} \in H_{i+1}K_{j+1}$ and that $[h_{i+1}, m_{new}] \equiv x_{new} \pmod{K_{j+1}}$. We know that $x, \overline{x} \in H_{i+1}$, since $[h_{i+1}, K] \subset H_{i+1}$. H_{i+1} is normalized by M, so

$$x_{new} = (\overline{x}\,\overline{m}^{\,-1})^{\alpha} x\overline{m}^{\,\alpha} \in H_{i+1}(\overline{m}^{\,-1})^{\alpha} x\overline{m}^{\alpha} = H_{i+1}.$$

Let $h = h_{i+1}$, $\phi = \theta(m)$, $\overline{\phi} = \theta(\overline{m})$. To show that $[h, m_{new}] \equiv x_{new} \pmod{K_{j+1}}$, we may assume that $K_{j+1} = 1$. Then $k = k_{j+1} \in Z(K)$, $[h, m] = xk^{\phi}$, $[h, \overline{m}] = \overline{x}k^{\overline{\phi}}$, so $h^{-1}mh = xm^{-1}k^{\phi}$, $h^{-1}\overline{m}h = x\overline{m}^{\,-1}k^{\overline{\phi}}$, so

$$[h, m\overline{m}^{\,\alpha}] = h^{-1}(m\overline{m}^{\,\alpha})^{-1}hm\overline{m}^{\,\alpha} = h^{-1}\overline{m}^{\,-\alpha}hh^{-1}\overline{m}^{\,-1}hm\overline{m}^{\,\alpha}$$

$$= (x\overline{m}^{\,-1})^{\alpha}k^{\alpha\overline{\phi}}xm^{-1}k^{\phi}m\overline{m}^{\,\alpha} = (x\overline{m}^{\,-1})^{\alpha}xm^{-1}m\overline{m}^{\,\alpha}k^{\alpha\overline{\phi}+\phi} = (x\overline{m}^{\,-1})^{\alpha}x\overline{m}^{\,\alpha},$$

as stated, since $\alpha\overline{\phi} + \phi = 0$ from the definition of α.

Implementation

Figure 8 shows the details of the updating algorithm. Here we use the same representation for both M and H that is based on the structure of the ambient group K, that is, on the sequence corresponding to the direct product of the Sylow p-subgroups of the transitive constituents. Note that we never need elements of this group, the only place where such an element appears in the algorithm is the lines

```
        Compute $\phi(r) \in GF(p)$ with
        $x_r^{-1}[h_{i+1}, m_r] \equiv k_{j+1}^{\phi(r)}$ mod $K_{j+1}$,
```

but the only thing we are interested in here is the exponent of k_{j+1}, which we can get by mapping $x_r^{-1}[h_{i+1}, m_r]$ to its vector representation and assuming that we chose k_{j+1} such that its vector representation has a 1 as the leading coefficient. Since we are using this same exponent for h_{j+1} in the case $H_j \neq H_{j+1}$, when computing the strong generating sequence for H, we make sure that all generators map to vectors with leading coefficients 1. To make the computation of the α's simpler, when we found ϕ_s, we compute $\beta \equiv 1/\alpha \pmod{p}$ and use m_s^{β} in place of m_s and 1 in place of ϕ_s in further computations.

It is also worth noting that if for some j, both $H_j = H_{j+1}$ and $M \cap K_j = M \cap K_{j+1}$, then we can skip that j in the main for loop, because then all ϕ's are necessarily zeros. This condition can easily be checked by checking whether $m_{j+1} = h_{j+1} = 1$.

Complexity of the Normalizer Algorithm

The normalizer algorithm consists of three nested **for** loops, each iterated at most t times. In terms of n, the size of the permutation domain, t is $O(n)$. The operations indicated in the innermost loops are permutation multiplications, raising permutations to powers bounded by p, and computation of leading coefficients (the ϕ's). Each of these operations can be carried out in time linear in the size of the permutation domain. For permutation multiplications, this is trivial. For the powers, to compute g^k we can start computing ω^{g^k} in time $O(k) = O(n)$, then we can get the images of the rest of the points in the orbit $\omega^{\langle g \rangle}$ using the identity $(\omega^g)^{g^k} = (\omega^{g^k})^g$ in constant time per every new point. Since the nontrivial orbits are of size at least p, the total time spent on this for all orbits is $O(n)$. Finally, to compute the ϕ's amounts to finding the vector representation of the permutation (constant time per coordinate) and then taking the inner product of that vector (of length $O(n)$) with another precomputed vector (a column of the change-of-basis matrix **B**). All of this is linear in n. So the overall running time of the algorithm is $O(n^4)$.

Intersection of Subgroups of a Nilpotent Group

The Problem

Given $K \leq Sym(\Omega)$, K nilpotent, and $H \leq K$, $G \leq K$. Find $G \cap H$. Here, again, we can reduce the problem to finding intersections of subgroups of a p-group.

```
{input:  A strong generating sequence $m_1, \ldots, m_t$ for $N_G(H_{i+1})$.}
{output:  A strong generating sequence for $N_G(H_i)$.}

begin
{Initialize.}
for $r := 1$ to $t$ do
    $x_r := 1 \in H_{i+1}$
for $j := i + 1$ to $t - 1$ do
    {$(m_1, \ldots, m_t)$ is a strong generating sequence for $M_j$,
     $x_1, \ldots, x_t \in H_{i+1}$, $x_r = 1$ for $j \le r$,
     and $[h_{i+1}, m_r] \equiv x_r \bmod K_j$ for $r = 1, \ldots, t$}
    for $r := 1$ to $j$ do
        Compute $\phi(r) \in GF(p)$ with
        $x_r^{-1}[h_{i+1}, m_r] \equiv k_{j+1}^{\phi(r)} \bmod K_{j+1}$
    if $H_j = H_{j+1}$ then
      if $\phi(r) \ne 0$ for some $r$ then
        $s := \max\{r \colon \phi(r) \ne 0\}$
        for $r := 1$ to $s - 1$ do
            Solve $\phi(s)\alpha(r) + \phi(r) = 0$ for $\alpha(r) \in GF(p)$
            $m_r := m_r m_s^{\alpha(r)}$
            $x_r := (x_s m_s^{-1})^{\alpha(r)} x_r m_s^{\alpha(r)}$
        $m_s := 1$
        $x_s := 1$
    else {$H_j \ne H_{j+1}$}
      for $r := 1$ to $j$ do
          $x_r := x_r h_{j+1}^{\phi(r)}$
    {$(m_1, \ldots, m_t)$ is a strong generating sequence for $M_{j+1}$,
     $x_1, \ldots, x_t \in H_{i+1}$, $x_r = 1$ for $j + 1 \le r$,
     and $[h_{i+1}, m_r] \equiv x_r \bmod K_{j+1}$ for $r = 1, \ldots, t$}
return $(m_1, \ldots, m_t)$
end.
```

Figure 8: Normalizer update from $N_G(H_{i+1})$ to $N_G(H_i)$.

The Algorithm

The algorithm for the intersection problem builds on the same principle as the normalizer algorithm, using the same data structure and setup. We will find, successively, the intersection of G and HK_i. For $i = 0$ the intersection is evidently G, and for $i = t$ it is $G \cap H$. It is also clear that $G \cap HK_{i+1} = (G \cap HK_i) \cap HK_{i+1}$, so at each step we can use the result of the previous one. Another observation is that $|G \cap HK_i : G \cap HK_{i+1}|$ is either 1 or p, so we can use the maximal subgroup algorithm for cutting down $G \cap HK_{i-1}$. Figure 9 shows the details. Similarly to the normalizer update algorithm, we will call the current intersection M, which we represent by a strong generating sequence, $(m_1, \ldots, m_t)$ and for each m_r in this generating sequence we maintain a permutation $x_r \in H$ such that $x_r m_r \in K_{i-1}$. If for some i not all the $x_r m_r$'s are in K_i and H does not cover K_{i-1}/K_i then we have to cut down M. Again, similarly to the normalizer update, we select $\overline{m} = m_s$, and compute ϕ_r for m_r and $\overline{\phi}$ for $\overline{m}$ such that $x_r m_r \equiv k_{i+1}^{\phi_r}$ and $\overline{x}\,\overline{m} \equiv k_{i+1}^{\overline{\phi}}$, then we compute α_r such that $\alpha_r \overline{\phi} + \phi_r = 0$, and update m_r and x_r. For the verification of the update step we can again assume that $K_{i+1} = 1$ and therefore $k_{i+1} \in Z(K)$. Then we can write $x_r m_r = k_{i+1}^{\phi_r}$, i.e. $x_r = m_r^{-1} k_{i+1}^{\phi_r}$, similarly $\overline{x} = \overline{m}^{-1} k_{i+1}^{\overline{\phi}}$. Raising both sides of the latter equation to the αth power and then multiplying it with the previous equation we get $\overline{x}^{\alpha_r} x_r = (\overline{m}^{\,-1} k_{i+1}^{\overline{\phi}})^{\alpha_r} m_r^{-1} k_{i+1}^{\phi_r} = \overline{m}^{\,-\alpha_r} m_r^{-1} k_{i+1}^{\alpha_r \overline{\phi} + \phi_r} = \overline{m}^{\,-\alpha_r} m_r^{-1}$, so $\overline{x}^{\alpha_r} x_r m_r \overline{m}^{\alpha_r} = 1$, showing that the updates of the x_r's and m_r's are correct.

Complexity

Here we have one less for loop than we had in the normalizer algorithm, otherwise the organization is the same, so the complexity is $O(n^3)$ after the setup time.

```
{input:  strong generating sequences for $G$ and $H$
         $(m_1, \ldots, m_t)$ and $(h_1, \ldots, h_t)$, respectively}
{output: a strong generating sequence for $G \cap H$.}
begin
{Initialize.}
for $r := 1$ to $t$ do
    $x_r := 1 \in H$
    $\phi(r) := 0$
for $i := 0$ to $t - 1$ do
    {$(m_1, \ldots, m_t)$ is a strong generating sequence for $G \cap (HK_i)$,
     $x_r \in H$ and $x_r m_r \in K_i$ for $r = 1, \ldots, t$}
    for $r := 1$ to $t$ and $m_r \neq 1$ do
        Compute $\phi(r) \in GF(p)$ with $x_r m_r \equiv k_{i+1}^{\phi(r)} \bmod K_{i+1}$
    if $h_{i+1} = 1$ then
      if $\phi(r) \neq 0$ for some $r$ then
        $s := \max\{r : \phi(r) \neq 0\}$
        for $r := 1$ to $s - 1$ do
            Solve $\phi(s)\alpha(r) + \phi(r) = 0$ for $\alpha(r) \in GF(p)$
            $m_r := m_r m_s^{\alpha(r)}$
            $x_r := x_s^{\alpha(r)} x_r$
        $m_s := 1$
        $x_s := 1$
        $\phi(s) := 0$
    else {$HK_i = HK_{i+1}$}
      for $r := 1$ to $t$ and $m_r \neq 1$ do
          $x_r := h_{i+1}^{-\phi(r)} x_r$
return $(m_1, \ldots, m_t)$
end.
```

Figure 9: Subgroup Intersection Algorithm

Centralizer in a Nilpotent Group

Element Centralizer Problem

Given $K \leq Sym(\Omega)$, K nilpotent, and $h \in K$, $G \leq K$. Find $C_G(h) = \{g \in G : gh = hg\}$. The reduction to the p-group case applies here, too.

The Algorithm for Element Centralizer

The main idea here is to compute $C_G(hK_j/K_j)$ for $j = 0, 1, \ldots, t$, or equivalently to find a subgroup chain $G = G_0 \geq G_1 \geq \cdots \geq G_t$ such that $G_j = \{g \in G : [h, g] \in K_j\}$. Clearly, $G_0 = G$ and $G_t = G_{t-1} = C_G(h)$. Also $|G_j : G_{j+1}| \in \{1, p\}$. To prove this, consider $H = \langle h \rangle$ and let i be such that $h \in K_i \setminus K_{i+1}$. Using the notation $H_j = H \cap K_j$, we get $H_i = H$. We use again M as a variable that changes from G_j to G_{j+1} in each iteration of the outermost loop. (M now centralizes HK_j/K_j, therefore normalizes $H_iK_j = HK_j$.) We can apply the results from the normalizer case that M acts on the 2-dimensional vector space $H_iK_j/H_{i+1}K_{j+1} = hK_j/K_{j+1}$ and here we are also looking for the kernel of this action. As we saw, this kernel is the same as the kernel of the homomorphism θ, where $\theta(g)$ is defined by $[h, g] \in k_{j+1}^{\theta(g)}K_{j+1}$. Here we don't need the x's, because for all elements $m \in M$, $[h, m] \in K_j$ already. The algorithm is shown on Figure 10. The fact that θ is a homomorphism justifies the modification step for the m_k's, and we don't need to check m_k for $k > j$ because in that case $[h, m_k] \in K_{k+1} \leq K_{j+1}$. Also, we can start j from i, which can be determined by examining the cycle structure of h to get to the level that it does not stabilize and then computing its vector representation corresponding to that level and finding the smallest K-invariant subspace to which it belongs.

```
{input:  $a$ strong generating sequence $m_1, \ldots, m_t$ for $G \leq K$, and
         an element $h$ of $K$.}
{output: $a$ strong generating sequence for $C_G(h)$.}

begin
if $h \in K_{t-1}$ then return $(m_1, \ldots, m_L)$
for $j := 1$ to $t-1$ do
    {$(m_1, \ldots, m_t)$ is a strong generating sequence for $C_G(hK_j/K_j)$}
    for $r := 1$ to $j$ and $m_r \neq 1$ do
        Compute $\phi(r) \in GF(p)$ with $[h, m_r] \equiv k_{j+1}^{\phi(r)} \bmod K_{j+1}$
    if $\phi(r) \neq 0$ for some $r$ then
        $s := \max\{r : \phi(r) \neq 0\}$
        for $r := 1$ to $s-1$ do
            Solve $\alpha(r)\phi(s) + \phi(r) = 0$ for $\alpha(r) \in GF(p)$
            $m_r := m_r m_s^{\alpha(r)}$
        $m_s := 1$
return $(m_1, \ldots, m_L)$
end.
```

Figure 10: Element Centralizer Algorithm

Subgroup Centralizer Problem

Given $K \leq Sym(\Omega)$, K nilpotent, and $H \leq K$, $G \leq K$. Find $C_G(H) = \{g \in G : gh = hg, \ \ \forall h \in H\}$.

This reduces to the element centralizer problem, simply centralize the generators of H by calls to the element centralizer, one at a time.

Complexity

Similarly to the intersection algorithm, the element centralizer algorithm runs in $O(n^3)$ time after the setup, therefore the subgroup centralizer algorithm runs in time $O(sn^3)$, where s is the number of generators for H.

CHAPTER VIII

THE STRUCTURE OF SOLVABLE PERMUTATION GROUPS

In this chapter we will see how one can take advantage of the well-known structure of solvable permutation groups to obtain a normal series with elementary abelian factors, together with homomorphisms of the factors into vector spaces over $GF(p_i)$, where p_i are the primes dividing the order of the group in question.

First we recall a standard embedding of a transitive but imprimitive permutation group into a wreath product (see, e.g. [Ca], Proposition 3.1.). We will use the notation established in the definition of wreath products in Chapter III.

Lemma 8.1

Let $G \leq Sym(\Omega)$ be a transitive group and let $\mathbf{\Delta} = (\Delta_1, \ldots, \Delta_t)$ be a block system for G. Let $H = G^{\Delta_1}_{\{\Delta_1\}}$. Then G can be embedded into $\tilde{G} = H \wr G^{\mathbf{\Delta}}$ and Ω can be identified with $\Delta_1 \times \mathbf{\Delta}$ in such a way that G is a subgroup of $\tilde{G}$.

Proof

Proof: We describe the identification of $\Delta_1 \times \mathbf{\Delta}$ with Ω and show that the elements of G are among the elements of $\tilde{G}$ (viewed as a permutation group on Ω). For each $j = 1, \ldots, t$ let us fix $x_j \in G$ such that $\Delta_1^{x_j} = \Delta_j$. Let us identify $(\delta, \Delta_j) \in \Delta_1 \times \mathbf{\Delta}$ with $\delta^{x_j} \in \Omega$, for $\delta \in \Delta_1$, $j \in \{1, \ldots, t\}$. Then, with this identification $\tilde{G}$ acts on Ω. We will show that for each element $g \in G$ there is an element $\tilde{g} = (h_1(g), \ldots, h_t(g), \bar{g})$ of $\tilde{G}$ such that $\tilde{g}$ acts on

Ω as g. For $g \in G$ and $j \in \{1, \ldots, t\}$ we define j^g so that $\Delta_j^g = \Delta_{j^g}$. For $j = 1, \ldots, t$ let $h_j(g) = x_j g x_{j^g}^{-1} |_{\Delta_1}$. It is easy to check that $\Delta_1^{x_j g x_{j^g}^{-1}} = \Delta_1$, so $h_j(g) \in H$. Let $\overline{g}$ be the image of g by the action of G on Δ. Then $(\delta, \Delta_j)^{\widetilde{g}} = (\delta^{h_j(g)}, \Delta_{j^g})$. Now (δ, Δ_j) is identified with $\omega = \delta^{x_j} \in \Omega$ and $(\delta^{h_j(g)}, \Delta_{j^g})$ is identified with $(\delta^{h_j(g)})^{x_{j^g}} = (\delta^{x_j g x_{j^g}^{-1}})^{x_{j^g}} = \omega^g$, so indeed, g acts on Ω the same way as $\widetilde{g}$. $\square$

Lemma 8.2

With the notation in Lemma 8.1, let N be a normal subgroup of H. Then $\overline{N} = \{(n_1, \ldots, n_t, 1) \in \widetilde{G} : n_1, \ldots, n_t \in N\}$ is a normal subgroup of $\widetilde{G}$.

Proof

We will show that for $\overline{n} = (n_1, \ldots, n_t, 1) \in \overline{N}$ and $\widetilde{g} = (h_1, \ldots, h_t, \overline{g}) \in \widetilde{G}$, $\widetilde{g}\overline{n}\widetilde{g}^{-1} \in \overline{N}$. It is easy to check that $\widetilde{g}^{-1} = (h_{1^{g^{-1}}}, \ldots, h_{t^{g^{-1}}}, \overline{g}^{-1})$. Then, we just check that $\widetilde{g}\overline{n}\widetilde{g}^{-1} = (n_{1^g}^{h_1^{-1}}, \ldots, n_{t^g}^{h_t^{-1}}, 1)$. The right-hand side is clearly in $\overline{N}$, since $n_j^h \in N$ for any $h \in H$ by the normality of N in H. $\square$

It is obvious from the definition of $\overline{N}$ that it is isomorphic with the direct product of t copies of N.

In what follows in this chapter, all groups mentioned are assumed to be solvable permutation groups, unless stated otherwise.

The following lemma has been known ever since Galois (see, e.g. [Hu], 3.2. Satz).

Lemma 8.3

Let G be a primitive solvable group. Then G has a unique minimal normal subgroup A, which is elementary abelian and regular, and G is the semidirect product of A and a point stabilizer subgroup. $\square$

As we have seen earlier, it is enough to show how to deal with the transitive case. For the sake of easier description we define an elementary abelian normal series with vector space representations of the factors as follows:

Definition 8.1

Let G be a finite solvable group.

We call the pair $((G_0, \ldots, G_m), (\phi_1, \ldots, \phi_m))$ an elementary abelian structure for G if the following hold:

a.) $G_0 = G$,

b.) $G_i \triangleleft G$, for $i = 1, \ldots, m$,

c.) G_{i-1}/G_i is an elementary abelian p_i-group,

d.) $\phi_i : G_{i-1} \to GF(p_i)^{d_i}$ is a homomorphism with kernel G_i for $i = 1, \ldots, m$.

□

Here we do not require that the ϕ_i's be nontrivial, i.e. we allow for $G_{i-1} = G_i$.

In the rest of the chapter we will see how one can recursively build an elementary abelian structure for the transitive solvable group G. From this, just like in the case of p-groups, one can build the structure for arbitrary solvable permutation groups.

In the base case, when G is the trivial group, we let $m = 0$, so the series of homomorphisms is empty.

If G is a primitive group, by Lemma 8.3, G is a semidirect product of its unique minimal normal subgroup A, and a point stabilizer subgroup H. Therefore, H is a homomorphic image of G, say $\psi : G \to H$ is a homomorphism with kernel A. So if H has an elementary abelian structure, $((H_0, \ldots, H_m), (\phi_1, \ldots, \phi_m))$, then

$((\psi^{-1}(H_0), \ldots, \psi^{-1}(H_m), A), (\phi_1 \circ \psi, \ldots, \phi_m \circ \psi, \pi))$ will be an elementary abelian series for G, where π is any homomorphism mapping A to $GF(p)^d$, where $|A| = p^d$.

Let now $G \leq Sym(\Omega)$ be transitive, but imprimitive. Since G is imprimitive, there is a system of maximal blocks, $(\Delta_1, \ldots, \Delta_t)$, on which G acts primitively. From this action we get the beginning ("head") of the elementary abelian structure for G. Now we describe how to get the "tail" part of the structure. Let K be the kernel of the above mentioned action. Let $H = G^{\Delta_1}_{\{\Delta_1\}}$, i.e. the transitive constituent on the points of Δ_1 of the subgroup of G that leaves Δ_1 invariant. Let $((H_0, \ldots, H_m), (\phi_1, \ldots, \phi_m))$ be an elementary abelian structure for H. Let $x_1, \ldots, x_t$ be as in Lemma 8.1 and let g be an element of K. Then g leaves all blocks invariant, therefore, still using the notation of Lemma 8.1 and Lemma 8.2, $\tilde{g}$ is of the form $\tilde{g} = (h_1(g), \ldots, h_t(g), 1)$. We define

$$G_i = \{g \in K : h_j(g) \in H_i, j = 1, \ldots, t\}.$$

Then $G_i = \overline{H_i} \cap G$ is normal in $G = \tilde{G} \cap G$. Let $\Psi : K \to H^t = H \times \ldots \times H$ (t direct factors) be the map that maps $(h_1, \ldots, h_t, 1) \in K$ to $(h_1, \ldots, h_t) \in H^t$. Then Ψ is an embedding and $\Psi(G_i) \leq H_i^t$. So G_{i-1}/G_i is isomorphic with a subgroup of $H_{i-1}^t/H_i^t \cong (H_{i-1}/H_i)^t$ and therefore G_{i-1}/G_i is an elementary abelian p_i-group. Define $\psi_i = (\phi_i \times \cdots \times \phi_i) \circ \Psi : G_{i-1} \to GF(p_i)^{td_i}$, then $((G_0, \ldots, G_m), (\psi_1, \ldots, \psi_m))$ can be used as the "tail" of the elementary abelian series for G.

CHAPTER IX

DATA STRUCTURE FOR SOLVABLE PERMUTATION GROUPS

We saw in the previous chapter that for solvable permutation groups G there is a recursively computable structure $((G_0, \ldots, G_m), (\phi_1, \ldots, \phi_m))$ where the factors G_{i-1}/G_i are elementary abelian and $\phi_i : G_{i-1} \to GF(p_i)^{p_i^{d_i}}$ are homomorphisms with kernel G_i. In this chapter we will see a description of a data structure and algorithms for computing a strong generating system for the series of normal subgroups G_i, together with the computation of the maps ϕ_i. We will also see that coordinates of the vector $\phi_i(g)$ for $g \in G_{i-1}$ are computed quite naturally from the permutation action of g (on a possibly extended domain).

In the previous chapter we used preimages to describe some of the subgroups in the normal series, here we will see that in practice we will never have to compute these preimages, it will be enough to be able to compute the maps themselves. The procedure we are using also eliminates the trivial steps in the series.

The Data Structure

Our data structure for the solvable permutation group G will consist of a series of elements of G, $(g_{11}, \ldots, g_{1d_1}, \ldots, g_{m1}, \ldots, g_{md_m})$, and a series of maps, $(\phi_1, \ldots, \phi_m)$ such that if we define $G_i = \langle g_{(i+1)1}, \ldots, g_{(i+1)d_{i+1}}, \ldots, g_{m1}, \ldots, g_{md_m} \rangle$, for $i = 0, \ldots, m$ (note that G_m is generated by the empty set, so it is the trivial group), then

a.) $G_0 = G$,

b.) $G_i \triangleleft G$, for $i = 1, \ldots, m$,

c.) $\phi_i : G_{i-1} \rightarrow GF(p_i)^{d_i}$ is a homomorphism with kernel G_i, for $i = 1, \ldots m$,

d.) $\phi_i(g_{ik}) = \mathbf{e}_k \in GF(p_i)^{d_i}$, where $\mathbf{e}_k$ denotes the vector with all 0 coordinates with the exception of the kth coordinate which is 1.

We start building the data structure with the computation of the maps, just as in the p-group case. For an intransitive group, the maps for the transitive constituents are computed exactly the same way as for p-groups. For a transitive but imprimitive group, first we compute a system of maximal blocks, map the generators to act on the block indices, then compute Schreier generators for the action on the first block. We do not have to compute those generators as permutations on the whole domain, we can restrict ourselves to compute the images of the points in the first block, Δ_1, only, just like in the case of p-groups. For the map that was denoted by Ψ in the previous chapter, seemingly we need t permutations, $x_1, \ldots, x_t$, but here, again, we only need the images of the points in Δ_1 by each x_j, and the images of the points in each block Δ_j by x_j^{-1}.

Note that this way we have the same number of generators for the primitive part, and although the number of Schreier generators for the recursive call can be t times the number of original generators, the size of the domain on which these operate has shrunk by the same factor, so the total input size for the recursive call did not increase.

The Primitive Case

The really interesting part of the computation is the case of primitive $G < Sym(\Omega)$, $|\Omega| = p^d$, in particular, how we compute (generators for) the elementary abelian normal subgroup A. The idea of the algorithm for this comes from Sims [Si2]. We want to go down on the derived series of G until we reach the last subgroup on it, which is abelian. In our case (for a primitive solvable group), it is a minimal normal subgroup, so it is necessarily generated by one normal generator (i.e. $A = \langle a \rangle^G$ for some (any) $1 \neq a \in A$). The algorithm finds such a normal generator and enough conjugates of it to generate A. This is done in stages, each stage corresponding to a smaller member of the derived series than the previous stage. At each stage, we start with an element of G that is obtained as a nontrivial commutator of two members from the previous stage, therefore is known to be further down in the derived series (we start with one of the original generators in the first phase). In each phase, we are growing an abelian group, generated by the starting element of that stage and its conjugates. If we found a new conjugate that does not commute with one of the generators of the group that we are growing, we start a new stage with the commutator of these two elements. When we reach a normally closed (in G) abelian subgroup, we have (linearly independent generators for) A. The algorithm for this computation is shown in Figure 11. (EANSG stands for elementary abelian normal subgroup.) Since we add a new element to T only if it makes the group $H = \langle T \rangle$ larger, the number of elements that we consider in the while loop is less than $|S| \log |G|$. In order to be able to do membership test in H, we can maintain a point stabilizer chain for it. For this, we can use Sims's observation in [Si2] that updating a point stabilizer chain for H with a new generator h that normalizes H requires a single pass down the chain to get a point stabilizer chain for $\langle H, h \rangle$. We have this condition, since when we add a new member y to T, y commutes with all the elements in T, therefore it commutes with $H = \langle T \rangle$.

```
EANSG($S, g$)
{input: $G$, a solvable primitive permutation group
    on $\Omega$, given by a set of generators $S$. $|\Omega| = p^d$
    and $1 \neq g \in G$.}
{output: $A = \langle T \rangle$, the unique minimal normal
    subgroup of $G$}

begin
$Y := \{g\}$
$T := \emptyset$
while $Y \neq \emptyset$
    Let $y \in Y$
    $Y := Y \setminus \{y\}$
    for all $u \in T$
        $z := [u, y]$
        if $z \neq 1$ then
          {$z$ is at least one step further
          down in the derived series of $G$}
          return EANSG($S, z$)
    {$y$ commutes with $\langle T \rangle$ }
    if $y \notin \langle T \rangle$ then
        $T := T \cup \{y\}$
        $Y := Y \cup \{y^s : s \in S\}$
return $T$
end.
```

Figure 11: Algorithm for computing generators for the minimal normal subgroup of a primitive solvable group

This procedure will necessarily provide d generators for A, each moving all points, cyclically in batches of p. The group A is regular, since it is a normal subgroup of a primitive group, therefore it is transitive, and an abelian transitive group is regular. Furthermore, if we fix a point $\omega \in \Omega$, every $g \in G$ can be uniqely written as $g = g_\omega a$, where $g_\omega \in G_\omega$ and a is the unique element of A for which $\omega^g = \omega^a$. The map $\sigma : G \to G_\omega$ for which $\sigma(g) = g_\omega$ is a homomorphism, because there is exactly one element of G_ω in each coset of A in G. (Suppose h_1 and h_2 are in the same coset and both fix ω, then $h_1 h_2^{-1} \in A$, but then $h_1 h_2^{-1}$ is the unique element of A that fixes ω, that is, the identity, so $h_1 = h_2$.) So G_ω is isomorphic to G/A.

Using these generators we can map the points of Ω to the set $\{0, \ldots, p^d - 1\}$, such that the homomorphism σ can be computed quite easily. First, since A is a vector space and the d generators $a_1, \ldots, a_d$ constitute a basis for it, each element $a \in A$ can be written uniquely as $a = a_1^{e_1} a_2^{e_2} \ldots a_d^{e_d}$, where $e_1, \ldots, e_d \in \{0, \ldots, p-1\}$. Now if we regard $e_1 e_2 \ldots e_d$ as a p-ary number, and we assign this number to ω^a, calling this map ν, then $\nu(\alpha^{g_\omega}) = \nu(\alpha) \ominus \nu(\omega^g)$, where $\ominus$ means modulo p subtraction at each p-ary digit. The renumbering process is very straightforward. First, we set $\nu(\omega) = 0$. Then we take each generator, starting from the end (i.e. with a_d), and for each already numbered point α, if $\nu(\alpha^{a_j})$ has not been set yet, we set $\nu(\alpha^{a_j}) = \nu(\alpha) + p^{d-j+1}$

Now we only have to tell how we compute the vectors for an element $a \in A$. This, with the above map ν in hand, is really easy, we simply take the p-ary digits of $\nu(a)$ as the coordinates of a vector in $GF(p)^d$.

Computing the Strong Generating Sequence

With this, we have set up the homomorphisms that correspond to the elementary abelian structure described in the previous chapter. (Note, that the maps corresponding to the homomorphisms usually admit a broader domain than just the subgroups of the elementary abelian structures that are encountered during the computations.) The next thing to do is to compute the sequence of strong generators. The algorithm for doing this is similar to the one that we saw for the p-groups, except that in this case, there might be several "new" strong generators corresponding to each subgroup G_i in the elementary abelian structure, namely as many as the dimension of G_i/G_{i+1} is (this can also be 0). To see how the algorithm works, imagine that we have the strong generating system, our task is only to prove that it is a good one, i.e. $G_{i-1} = \langle g_{i1}, \ldots, g_{id_i}, \ldots, g_{m1}, \ldots, g_{md_m} \rangle$, $G_i \triangleleft G$ for $i = 0, \ldots m-1$, G_{i-1}/G_i is elementary abelian, i.e. a vector space and $dim(G_{i-1}/G_i) = d_i$. For this, we have to prove that all commutators $[g_{ia}, g_{ib}]$, $1 \leq a < b \leq d_i$ and all p_i-powers $g_{ia}^{p_i}$ for $a = 1, \ldots, d_i$ are in G_i, and all conjugates of g_{ia}^g, $a = 1, \ldots, d_i$, $g \in S$, are in G_{i-1}.

Now, if we have built such a system, we can sift through it, i.e. decompose elements of G as products of elements of the strong generators. Also, if we take an element $h \in Sym(\Omega)$, we can decide, using the sifting process, whether $h \in G$. For this, before applying any homomorphism in the sifting process, we should first check whether the permutation to which we want to apply the homomorphism is in the domain of it. If it fails to be, $h \notin G$. We can draw the same conclusion if at a point in the sifting process we can map our permutation into a vector space, but the resulting vector is not a linear combination of the vectors corresponding to the strong generators at that level.

This sifting process can also be used to build the data structure, i.e. to compute the strong generators. To do this, we sift through a partially built data structure, and if something does not "sift through", we add that permutation to the list of strong generators at the level it "got stuck". In this case, we know that every permutation that shows up during the computation is in G, since we start with the generators of G and every newly computed element is either an inverse of a previous one or a product of two earlier elements. Therefore, in this case we can be sure that whenever we want to apply a map to a permutation h in the process, h is in the domain of the map. So the sifting can "get stuck" only for the second reason: at some level, we have to expand our basis, i.e. add the new permutation to the strong generators corresponding to that level. Whenever this happens, we sift further the commutators of the newly added member with all previous members of the strong generating system at that level, it's p_ith power, and its conjugates with all of the original generators of the group. At the end of this process, we have sifted through everything that we had to (either it sifted already through the partially built strong generating system, or the residue of it was added to the system) in order to prove that we have the strong generating system belonging to the elementary abelian structure. At the end of the process we perform Gaussian elimination at each level to get generators corresponding to the unit vectors. The outline of the algorithm is shown in Figure 12. The algorithm assumes that $sift(g, j, (G_1, \ldots, G_m), (\phi_1, \ldots, \phi_m))$ starts sifting g at level j and it returns a pair consisting of the residue of g and the level where it "got stuck".

Complexity

Sifting through one level means computing a vector representation at that level and then dividing out powers of the generators of the level. The first part takes $O(n \log n)$ time, while the second takes cnd time where c is a constant and d is the dimension of the level. There are $O(n \log n)$ levels, so a sift through the whole structure takes $O(n^2 \log^2 n)$ steps for the first part and a total of $O(n^2)$ for the second,

```
{input:  G, a solvable permutation group, given by the
         the generating set S
         ($(\phi_1, \ldots, \phi_m)$), a series of maps of an elementary abelian
         structure for G is already computed }
{output: a strong generating system for H}
begin
$T := emptystack()$
$(G_1, G_2, \ldots, G_m) = (\emptyset, \emptyset, \ldots, \emptyset)$
for all $z \in S$ do $push((z, 1), T)$
while $T$ is not empty do
        $(g, j) := pop(T)$
        $(g, i) := sift(g, j, (G_1, \ldots, G_m), (\phi_1, \ldots, \phi_m))$
        if $i \neq m + 1$ then
           for all $h \in G_i$ do
                   $push(([g, h], i + 1), T)$
           $push((g^{p_i}, i + 1), T)$
           for all $z \in S$ do
                   $push((g^z, i), T)$
        $G_i := G_i \cup \{g\}$
return $(G_1, G_2, \ldots, G_m)$
end.
```

Figure 12: Computing the strong generating sequence

because the dimensions add up to $O(n)$. We have to sift $O(n^2)$ commutators, $O(n)$ powers and $O(sn)$ conjugates, in addition to the s generators, a total of $O(sn + n^2)$ permutations. So the total running time of the algorithm is $O(sn^3 \log^2 n + n^4 \log^2 n)$.

CHAPTER X

SOME ALGORITHMS FOR SOLVABLE PERMUTATION GROUPS

As we pointed out earlier, if one has special algorithms for input with special properties, it is useful to have a fast way to test whether the input has the property. In the first part of this chapter we show a method for testing solvability which is in the general case faster than the usual ways (computing the derived series, or attempting to compute a polyciclic generating sequence – see [Si2])

In the second part of the chapter we illustrate the usefulness of the data structure of Chapter IX by showing an algorithm for computing Sylow p-subgroups of solvable groups. With minimal modification the algorithm can be applied to compute Hall subgroups, too.

Recognition of Solvability

In this section we will prove the following.

Proposition 10.1

Let $G = \langle S \rangle \leq Sym(\Omega)$, $|S| = s$, $|\Omega| = n$. Then we can decide whether G is solvable in time $O(sn^2)$. $\square$

We start with a lemma about properties that are inherited by results of certain operations. When we talk about permutation groups in the lemma, we do not necessarily mean that the groups have the same permutation domain.

Lemma 10.1

Let $\mathcal{P}$ be a property for permutation groups and suppose that $\mathcal{P}$ is inherited by subgroups, homomorphic images, wreath products and direct products. Then a group G has the property $\mathcal{P}$ if and only if every primitive group constructed from G by the above operations has $\mathcal{P}$.

Proof

If G has $\mathcal{P}$ then so does every group constructed from G by the mentioned operations by inheritance. Conversely, using those operations, by the result expressed in Lemma VIII.1 and the obvious embedding of an intransitive permutation group into the direct product of its transitive constituents, we can construct primitive groups from which we can rebuild G using the permitted operations. So if all those primitive ingredients have $\mathcal{P}$, so does G. $\square$

By Lemma 10.1 we can reduce testing groups for a property $\mathcal{P}$ to testing primitive groups for $\mathcal{P}$, if property $\mathcal{P}$ satisfies the required inheritance condition. Now, if $\mathcal{P}$ is the property of being solvable, the condition is satisfied, so testing solvability reduces to testing solvability of primitive groups.

The method that we will use for testing primitive groups will attempt to compute a subgroup chain with abelian factors for the group we are testing. If the attempt is succesful, we report that the group is solvable. The next two lemmas provide us with theoretical results that enable us to abort this computation if we have computed a subgroup that is too big to be solvable, so we can report non-solvability in a timely manner.

By a result of Pálfy [Pá] and Wolf [Wo], solvable primitive groups are fairly small.

Lemma 10.2

Let G be a solvable primitive permutation group on n points. Then $|G| \leq 24^{-\frac{1}{3}} n^{1+c_0}$, where $c_0 = 2.243\ldots$. □

Lemma 10.2 provides an $O(\log n)$ upper bound on the length of any subgroup chain for a solvable group acting primitively on n points.

Dixon [Di] gives an upper bound on the length of the derived series of solvable groups. If we apply his result to solvable primitive groups, we get the following.

Lemma 10.3

Let G be a solvable primitive permutation group on $n = p^d$ points. Then the length of the derived series of G is not more than $1 + \frac{5}{2}(\log_3 d + 1)$.

□

This means that the length of the derived series of a primitive solvable group acting on n points is $O(\log \log n)$.

The algorithm for testing solvability of a primitive permutation group is a modification of Sims's algorithm [Si2]. The modified algorithm is shown on Figure 13 and Figure 14. We may assume that the group is acting on p^d points, since otherwise it could not be solvable (i.e. the test for solvability really starts with checking whether the domain of the primitive action has a prime-power number of elements).

The heart of the algorithm is the function **AbelianNormalSeriesPrim**, that has the following arguments: G is a primitive permutation group; $g \in G$, U is a strong generating sequence for the chain of G-normal subgroups $N_0 > N_1 > \cdots > N_t = 1$,

```
{input: $G$, a primitive permutation group on $\Omega$,
    given by a set of generators $S$. $|\Omega| = p^d$}
{output: true if $G$ is solvable, false otherwise}
begin
$U := ()$
$l := \lceil 24^{-\frac{1}{3}} n^{3.244} \rceil$
$r := 1 + \frac{5}{2}(\log_3 d + 1)$
for all $g \in S$
    if $g \notin \langle U \rangle$ then
      $U :=$ AbelianNormalSeriesPrim($S, g, U, r, l$)
      if $U$ =false then return false
return true
end.
```

Figure 13: Algorithm for testing solvability of a primitive group

where N_{i-1}/N_i is abelian. We assume that $g \notin N_0$. The other two parameters are used to abort the computation if there is evidence that G cannot be solvable: r gives an upper bound on the depth of recursion, while l is the maximum length of an increasing subgroup chain for solvable G. The function returns a strong generating sequence for $\langle g, U\rangle^G$. This sequence also has the property that no element of it is in the group generated by all subsequent elements of the sequence. Therefore the length of the sequence is a lower bound for the length of an increasing subgroup chain of G. The goal in **AbelianNormalSeriesPrim** is to find a set of generators for a normal subgroup of G that contains both g and $\langle U_0\rangle$. These generators are being stored in U and T. The generators in U will generate a normal subgroup of G, while the generators in T will commute modulo $\langle U\rangle$. In Y we collect conjugates of elements of T and eventually we prove that these are all in the group generated by T and U. To reach this goal we sometimes add elements to T (if a conjugate happens to be outside

```
**AbelianNormalSeriesPrim**$(S, g, U_0, r, l)$
{input: $G$, a primitive permutation group, given by a set of generators $S$,
    $1 \neq g \in G$,
    $U_0$, a strong generating sequence for a $G$-normal series of $\langle U_0 \rangle \triangleleft G$,
    $r$, an indicator of $g$'s position in the derived series of $G$
    $l$, an upper bound on the length of subgroup chains of $G$ if $G$ is solvable.}
{output: either false, in which case $G$ is not solvable,
    or a strong generating sequence for a $G$-normal series of $\langle U_0, g \rangle^G$.}

begin
if r=0 then return false
$Y := \{g\}$
$U := U_0$
$T := ()$ {an empty sequence}
while $Y \neq \emptyset$
{$\langle U \rangle \triangleleft G$, $\langle T, U \rangle / \langle U \rangle$ is abelian, $\{g\} \cup T^S \subset \langle T, U \rangle \cup Y \subset \langle U_0, g \rangle^G$}
    Let $y \in Y$
    $Y := Y \setminus \{y\}$
    $V := U$
    for all $u \in T$
        $z := [u, y]$
        if $z \notin U$ then
            {$z$ is at least one step further down
            in the derived series of $G$ than $g$}
            $U :=$AbelianNormalSeriesPrim$(S, z, U, r-1, l)$
            if $U =$ false then return false
    {$y$ commutes with $\langle T \rangle$ modulo $\langle U \rangle$ }
    if $U \neq V$ then
        $W := ()$
        for all $u \in T$
            if $u \notin \langle W, U \rangle$ then add $u$ to $V$ as its first element
        $T := W$
    if $y \notin \langle T, U \rangle$ then
        add $y$ to $T$ as its first element
        $Y := Y \cup \{y^s : s \in S\}$
    if $|T| + |U| > l$ then return false
return the concatenation of $T$ and $U$
end.
```

Figure 14: Algorithm AbelianNormalSeriesPrim

of $\langle T, U \rangle$) or increase U (by a recursive call, if a new perspective member of T does not commute with an element of T modulo $\langle U \rangle$).

Correctness

To prove the correctness of the algorithm we have to show that it returns **true** if and only if G is solvable. The algorithm returns **true** only if it computes a normal series for G with abelian factors, and therefore G is solvable in this case. Conversely, suppose that G is solvable. Then we have to prove that **AbelianNormalSeriesPrim** always returns a generating sequence for $\langle U, g \rangle$. For this we have to check that the algorithm never returns **false**. It is easy to see that $|U|+|T| < l$ if G is solvable, since when we add a new element to T, we increase the size of $\langle T, U \rangle$. By Lemma 10.2, we cannot do this more than l times. Similarly, in a solvable primitive group we cannot descend on the derived series more than $1 + \frac{5}{2}(\log_3 d + 1)$ times, by Lemma 10.3. Now we prove that what **AbelianNormalSeriesPrim** returns is indeed a strong generating sequence of a G-normal series with abelian factors. Let the derived series of G be $G = G_0 > G_1 > \cdots > G_m = 1$. For $h \in G$ we define $e(h) = \max\{i : h \in G_i\}$. We prove the correctness by induction on $e(g)$ (starting from $m-1$ and decreasing). If $e(g) = m-1$ then g commutes with all of its conjugates, so there will be no recursive calls, and therefore the first part of the loop invariant is trivially true. The other two are also very easily checkable. Now if $e(g) < m-1$ and there is a recursive call, then in that $e(z) > e(g)$, since z is a commutator of two conjugates of g, so by induction it returns a strong generating sequence for a normal subgroup of G. The statement starting with **if** $U \neq V$ is used to update T in case of a recursive call that increased $\langle U \rangle$. It restores the property of T that each element of it increases the size of the group generated by the other elements of T and the elements of U. The next

if statement adds y to T if necessary, and adds its conjugates to the elements that should be checked. If there are no more elements to check, we now that $\langle T, U\rangle$ is normal in G and since T consists of conjugates of g, $\langle T, U\rangle = \langle g, U\rangle^G$.

Complexity of Testing Solvability

First we argue about the complexity of testing whether a primitive group G is solvable. The crucial observation in the argument is that whenever **AbelianNormalSeriesPrim** is called recursively, both $|U|$ and the size of the group generated by U increased. This means that there will be no more than l calls altogether (for solvable groups, this comes from the theoretical bound on the size of the group (Lemma 10.2), and for non-solvable groups from the check of $|T| + |U|$ that forces to return **false** after at most l calls. Other than the recursive calls, there is a while loop that is executed as many times as there are elements in Y. As we put elements to Y only when we add a new generator to T (which increases the size of the group $\langle T, U\rangle$), this number is $O(s \log n)$. Within the loop, we do T membership tests in $\langle U\rangle$, possibly another T membership tests in $\langle W, U\rangle$ and possibly the same number of extensions of the group $\langle W, U\rangle$ by a normalizing element, followed by possibly one more such extension of $\langle T, U\rangle$. For these groups we keep a point-stabilizer series, with the help of which, all extensions and membership-tests can be done in $O(n \log n)$ time (since the length of the stabilizer-chain is $O(\log n)$. This means that without the recursive calls the time spent in **AbelianNormalSeriesPrim** is $O(sn \log^3 n)$. The algorithm shown on Figure 13 also calls **AbelianNormalSeriesPrim** only in case U is to be increased, so the total time spent in all invocations of **AbelianNormalSeriesPrim** is $O(sn \log^4 n)$. The rest of the algorithm is just s membership-tests in $\langle U\rangle$, therefore the total time for deciding whether a primitive group is solvable is $O(sn \log^4 n)$.

For transitive G, to find a system of maximal blocks (or decide that the group is primitive) takes $O(sn^2)$ time. If G is primitive, to test solvability takes further $O(snlog^4n)$ time. In the case of an imprimitive G, if the number of blocks is t, we reduce the problem to finding out whether the action on the blocks is solvable ($O(st \log^4 t)$) and recursively test whether the transitive group $G^{\Delta_1}_{\{\Delta_1\}}$ is solvable. For this we compute st generators for $G^{\Delta_1}_{\{\Delta_1\}}$, which permute n/t points. To find these generators takes linear time (similarly to the p-group case in Chapter VI). So if we denote by $T(s,n)$ the time required to find out whether a transitive group is solvable, we have the equation $T(s,n) \leq T(st, n/t) + O(sn^2)$, the solution of which is $T(s,n) = O(sn^2)$.

Finally, for intransitive groups the time required to reduce the problem to determine solvability of the transitive constituents is linear, with s staying the same and n linearly decreasing, so the total time is still $O(sn^2)$.

Finding Sylow Subgroups in Solvable Groups

In this section we present an algorithm to compute generators for a Sylow p-subgroup of the solvable group G. The algorithm needs to be changed only minimally to compute Hall subgroups. It has a similar abstract structure to the algorithm in [EW] and it is a variation of the one in [KLM]. It makes use of the vector spaces that naturally arise from the permutation structure and are part of our data structure.

In addition to the utilization of the above mentioned vector spaces, the algorithm also uses presentations of factors.

Let X be a set and let $\mathcal{F}(X)$ be the free group on X. Let G be a group and $N \triangleleft G$. Let $\phi : X \to G$, and let us denote the extension of ϕ to a homomorphism from $\mathcal{F}(X)$ to G by $\widehat{\phi}$. Let $\mathcal{R} \subset \mathcal{F}(X)$ and suppose that $\langle\phi(X)\rangle N = G$ and $\widehat{\phi}^{-1}(N) = \langle\mathcal{R}\rangle^{\mathcal{F}(X)}$.

Lemma 10.4

Under the above conditions, $\langle X|\mathcal{R}\rangle$ is a presentation for G/N.

Proof

Let $\pi : G \to G/N$ be the natural homomorphism.

Then $\pi \circ \widehat{\phi} : \mathcal{F}(X) \to G/N$ is an epimorphism with kernel $\langle \mathcal{R}\rangle^{\mathcal{F}(X)}$. □

Definition 10.1

Let X, G, N, $\mathcal{R}$ and ϕ be as above. Then we say that the presentation $\langle X|\mathcal{R}\rangle$ of G/N is realized via ϕ. We also say that $\Pi = \langle X, \phi|\mathcal{R}\rangle$ is a constructive presentation for G/N. □

Figure 15 shows the outline of an algorithm for finding a Sylow p-subgroup for the solvable group G (we describe the details in the following sections). This generic algorithm is similar to that in [Ma] (see also [KLM]). We input G by its elementary abelian structure, so $G = G_0 > G_1 > \cdots > G_t = 1$ is a normal series for G where G_{i-1}/G_i is an elementary abelian p_i-group ($i = 1, 2, \ldots, t$), and we have homomorphisms $\beta_i : G_{i-1} \to GF(p_i)^{d_i}$. We assume that we have constructive presentations for the factors G_{i-1}/G_i.

We start by finding the first i for which $p_i = p$. Let $r = \min\{i : p_i = p\}$. Then G_r still contains a Sylow p-subgroup of G. We also look for the largest G_i that is still a p-group, this may save some computation at the end. The real work is done in the last **while** loop in which we maintain the constructive presentation $\Pi = \langle X, \phi|\mathcal{R}\rangle$ for H/G_{i-1}, where $H = \langle \phi(X), G_{i-1}\rangle$.

The invariant for this last **while** loop is the following: $\langle\phi(X)\rangle G_{i-1}/G_{i-1}$ is a Sylow p-subgroup of G/G_{i-1} and Π is a constructive presentation for $\langle\phi(X)\rangle G_{i-1}/G_{i-1}$.

```
{input: $G$, a solvable permutation group, given by an elementary
    abelian structure $((G_0, G_1, \ldots, G_t), (\psi_1, \psi_2, \ldots, \psi_t))$,
    constructive presentations $\Pi_i = \langle X_i, \phi_i | \mathcal{R}_i \rangle$ for $G_{i-1}/G_i$, where
    $|G_{i-1}/G_i| = p_i^{d_i}$, for $i = 1, 2, \ldots, t$
    and a prime $p$.}
{output: A Sylow $p$-subgroup of $G$.}

begin
$i := 1$
while $i < t$ and $p_i \neq p$ do $i := i + 1$
if $i = t$ then return 1
while $t > i$ and $p_t = p$ do $t := t - 1$
if $i = t$ then return $G_{t-1}$
$\Pi = \langle X, \phi | \mathcal{R} \rangle := \langle \emptyset, \emptyset | \emptyset \rangle$
while $i \leq t$
    if $p_i = p$ then
      $\Pi :=$ExtendPresentation$(\Pi, \Pi_i)$
    else
      $\Pi :=$Complement$(\Pi, \Pi_i, \psi_i)$
    i:=i+1
return $\langle \phi(X), G_t \rangle$
end.
```

Figure 15: Sylow p-subgroup algorithm

When we first enter the loop, $i = r$, $H = G_{r-1}$ and the presentation is the empty presentation for $G_{r-1}/G_{r-1} = 1$. At each iteration we take one step down the chain, i.e. we want to update H to $\overline{H}$ such that $\overline{H}/G_{i+1}$ is a Sylow p-subgroup of G/G_{i+1}, we face one of two cases: (i) $p_i = p$ or (ii) $p_i = q \neq p$.

In case (i), H/G_i is a p-group, so we can take $\overline{H} = H$ and we have to compute a new constructive presentation for H/G_i from the constructive presentations of H/G_{i-1} and G_{i-1}/G_i. This includes adding new generators, computing new relators and extending the map ϕ to $\overline{\phi}$ to include the new generators (see the details below, in section "Extending the constructive presentation").

In case (ii), we will find $\overline{H} < H$ such that $\overline{H}/G_i \cong H/G_{i-1}$, we keep X and $\mathcal{R}$ from the constructive presentation, and we change ϕ to $\overline{\phi}$ such that $\overline{H}/G_i$ is realized via $\overline{\phi} : X \to \overline{H}$. In both cases, $\overline{H}/G_i$ is a Sylow p-subgroup of G/G_i and we have a presentation that is realized via $\overline{\phi}$. Of course, we can stop the process if G_i is already a p-group (see the next section).

<u>Finding Complements</u>

We will examine case (ii) of the previous paragraph in a more general situation. Let G be a group, $N \triangleleft G$ and $\langle X|\mathcal{R}\rangle$ a presentation for G/N realized via the map $\phi : X \to G$.

<u>Definition</u> 10.2

We say that $C < G$ is a complement of N in G, if $C \cap N = 1$ and $G = CN$.

□

We want to answer the question whether N has a complement in G and if so, find one. The following lemma provides a necessary and sufficient condition for the existence of a complement.

Lemma 10.5

Let G be a group, $N \triangleleft G$ and let $\langle X | \mathcal{R} \rangle$ be a presentation for G/N realized via ϕ. Then N has a complement C in G if and only if there exist a map $\theta : X \to G$ such that $\widehat{\theta}(\mathcal{R}) = 1$ and $\theta(x)^{-1}\phi(x) \in N$ for all $x \in X$.

Proof

Note that the normality of N implies that $\forall\ (x \in X)\ (\theta(x)^{-1}\phi(x) \in N)$ if and only if $\forall\ (\omega \in \mathcal{F}(X))\ (\widehat{\theta}(\omega)^{-1}\widehat{\phi}(\omega) \in N)$. This shows that $\langle X | \mathcal{R} \rangle$ is realized via θ, too. If C is a complement to N then every $g \in G$ can be uniquely written as $g = g_c g_n$ with $g_c \in C$ and $g_n \in N$. Let $\theta(x) = \phi(x)_c$, then θ satisfies the stated conditions. In the other direction, if we have θ with the stated conditions, then let $C = \widehat{\theta}(\mathcal{F}(X))$. We want to show that $C \cap N = 1$. Let $\omega \in \mathcal{F}(X)$ such that $\widehat{\theta}(\omega) \in N$. Then $\widehat{\phi}(\omega) \in N$ and therefore $\omega \in \langle \mathcal{R} \rangle^{\mathcal{F}(X)}$, so $\widehat{\theta}(\omega) = 1$. □

To get an algorithm with better complexity, we will need a slightly stronger statement. Suppose in addition that for some $Y \subset X$, $B = \langle \phi(Y) \rangle$ and B is a complement of N in BN and we are interested in the existence of a complement C of N in G such that $B < C$.

Lemma 10.6

Let G be a group, $N \triangleleft G$ and let $\langle X | \mathcal{R} \rangle$ be a presentation for G/N realized via ϕ. Let us suppose that for some $Y \subset X$, $B = \langle \phi(Y) \rangle$ and B is a complement of N in BN. Then N has a complement $C > B$ in G if and only if there exist a map $\theta : X \to G$ such that (i) $\widehat{\theta}(\mathcal{R}) = 1$, (ii) $\theta(y) = \phi(y)$ for all $y \in Y$ and (iii) $\theta(x)^{-1}\phi(x) \in N$ for all $x \in X$.

Proof

Suppose that such C exists. Then the θ given in the proof of Lemma 10.5 satisfies the additional condition. Conversely, if there exist θ with these conditions, $C = \widehat{\theta}(\mathcal{F}(X)) > \langle\theta(Y)\rangle = \langle\phi(Y)\rangle = B$. $\square$

Let ω be a word in the letters of $X = \{x_1, x_2, \ldots, x_k\}$. Let $\omega(\mathbf{c}) = \omega(c_1, c_2, \ldots, c_k)$ denote the word that we get from ω by substituting each occurrence of x_i by the expression c_i (and the occurrences of x_i^{-1} by c_i^{-1}). If the c_i's are group elements (e.g. permutations), we will use the same notation for the group element (permutation) that we get doing the multiplications and taking the inverses prescribed by ω.

By Lemma 10.5 a complement of N in G exists if and only if there are $n_1, \ldots, n_{|X|} \in N$ such that $\omega(\phi(x_1)n_1, \phi(x_2)n_2, \ldots, \phi(x_{|X|})n_{|X|}) = 1 \in G$ for all $\omega \in \mathcal{R} = \{\omega_1, \ldots, \omega_{|\mathcal{R}|}\}$.

If we write "unknowns", i.e. new symbols in place of the n_i's, we can express the condition in terms of solvability of a system of equations. Let $Y = \{y_1, y_2, \ldots, y_{|X|}\}$, such that $X \cap Y = \emptyset$. For each ω_j we define v_j, a word in $X \cup Y$, as $v_j = \omega_j(x_1y_1, x_2y_2, \ldots, x_{|X|}y_{|X|})$.

Now the condition for the existence of a complement becomes: there is a solution of the system of equations $v_j(\mathbf{g}, \mathbf{y}) = v_j(g_1, g_2, \ldots, g_{|X|}, y_1, y_2, \ldots, y_{|X|}) = 1$, $j = 1, 2, \ldots, |\mathcal{R}|$, where $g_i = \phi(x_i)$ and $y_1, \ldots, y_{|X|}$ are unknowns.

Kantor, Luks and Mark in [KLM] (Lemma 3.6) prove the following

Lemma 10.7

Let $N \triangleleft G$, N be abelian, $g \in G^{|X|}$ and let us use the notation $\mathbf{1} = (1, \ldots, 1) \in N^{|X|}$. Then the map $\Phi_v : N^{|X|} \to N$, defined by $\Phi_v(\mathbf{n}) =$

$v(\mathbf{g},\mathbf{1})^{-1}v(\mathbf{g},\mathbf{n})$ is a homomorphism. □

In terms of the Φ_v's, our equations take the form $\Phi_{v_j}(\mathbf{y}) = v_j(\mathbf{g},\mathbf{1})^{-1}$.

If N is elementary abelian, we can treat N as a vector space and the Φ_v's become linear transformations. So in this case we have to find a solution of a system of linear equations in order to find a complement. In fact, by solving the system, we find all possible complements (including the case when there is none). The algorithm for finding a complement in this case is shown on Figure 16.

In the case to which we apply this method for finding complements in the Sylow p-subgroup algorithm, N is an elementary abelian q-group (G_i/G_{i+1}), and in the role of G we have H/G_{i+1}. Since H/G_i is a p-group and $p \neq q$, we know that a complement exists and it is a Sylow p-subgroup of H/G_{i+1}, therefore we know that our system of equations has a solution. We only have to find one.

We set up the system of linear equations as follows. (This is similar to what is shown and demonstrated on a small example in [CNW].) From a basis of N, we form a basis of $N^{|X|}$, let this basis be $\mathbf{B} = (\mathbf{b}_1, \mathbf{b}_2, \ldots, \mathbf{b}_{d|X|})$, where d is the dimension of N. For each $j \in \{1, 2, \ldots, |\mathcal{R}|\}$ we compute $\Phi_{v_j}(\mathbf{b}_k) \in N$ ($k = 1, 2, \ldots, d|X|$), and interpret it as the kth column of the matrix T_j. Then T_j is the matrix of the linear transformation Φ_{v_j} in the basis $\mathbf{B}$. The right-hand side of the equations is the vector corresponding to $v_j(\mathbf{g},\mathbf{1})^{-1}$. This way we set up d equations for each relator.

In the rest of this section we show how the data structure presented in Chapter IX helps us to do the computations efficiently. In addition to what we have in the data structure we need representation of the constructive presentations.

We will represent the relators as straight-line programs. This will save us some in the evaluation of the relators. The term "straight-line program" in computational

```
{input: Constructive presentations $\Pi = \langle X, \phi | \mathcal{R} \rangle$ and $\Sigma = \langle Y, \psi | \mathcal{T} \rangle$ for $G/K$
    and $N/K$, respectively, where $N \triangleleft G$, $N/K$ is an elementary abelian $q$-group
    and $\beta : N \to GF(q)^d$ epimorphism with kernel $K$, such that the $k$-th
    component of $\beta(\psi(y_j)) = \delta_{kj}$ (Kronecker $\delta$)}
{output: A constructive presentation for a complement of $N/K$ in $G/K$. }

begin
$(u_1, \ldots, u_d) := \psi(Y)$
$(g_1, \ldots, g_{|X|}) := \phi(X)$
for $z := 1$ to $|X|$ do $h_z := g_z$
forall $\omega \in \mathcal{R}$ do
   $v_\omega = \Lambda_\omega(g_1, \ldots, g_{|X|}) - 1$
   $\mathbf{b}_\omega = \beta(v_\omega)$
   for $z := 1$ to $|X|$ do
       for $k = 1$ to $d$ do
           $h_z := g_z u_k$
           $D_\omega[d(z-1)+k] := \beta(v_\omega \Lambda_\omega(h_1, \ldots, h_{|X|}))$
       $h_z := g_z$
$D := \begin{bmatrix} D_1 \\ \vdots \\ D_{|\mathcal{R}|} \end{bmatrix}$ and $\mathbf{b} := \begin{bmatrix} \mathbf{b}_1 \\ \vdots \\ \mathbf{b}_{|\mathcal{R}|} \end{bmatrix}$
Let $\mathbf{e} = (e_1, \ldots, e_{d|X|})$ be a solution of $D\mathbf{x} = \mathbf{b}$ if that exists and $\mathbf{e} := \emptyset$ otherwise
bf if $\mathbf{e} = 0$ then return false
for $z := 1$ to $|X|$ do
   $n_z = u_1^{e_{(z-1)d+1}} u_2^{e_{(z-1)d+2}} \cdots u_d^{e_{zd}}$
Let $\overline{\phi}$ be defined by $\overline{\phi}(x_z) = g_z n_z$
return $\langle X, \overline{\phi} | \mathcal{R} \rangle$
end.
```

Figure 16: Complement algorithm

group theory (the theory of the so called black box groups) is usually used in a more restricted form than what we want to use it. We define a straight line program as follows.

Definition 10.3

Let $V = \{v_1, v_2, \ldots\}$ be a set of symbols and let Γ be another countably infinite set, disjont from V. We call Γ the set of operator symbols. Each $\gamma \in \Gamma$ has an arity $r(\gamma)$, and Γ contains a subset $\{\alpha_1, \alpha_2, \ldots\}$ with $r(\alpha_j) = 0$ for $j = 1, 2, \ldots$. We call the α_j's constant operators. A straight-line program is a sequence of assignment statements $\Lambda = (a_1, a_2, \ldots, a_k)$, where each statement a_i is of the form $v_i = \gamma_i(v_{i_1}, v_{i_2}, \ldots, v_{i_{r(\gamma_i)}})$, where $\gamma_i \in \Gamma$ and $i_1, i_2, \ldots, i_{r(\gamma_i)} < i$. □

If G is a group than we can interpret a straight-line program in G as follows. Let $(g_1, g_2, \ldots)$ be a sequence of elements of G and for each $\gamma \in \Gamma$ let $f_\gamma : G^{r(\gamma)} \to G$ be a function, such that $f_{\alpha_i}() = g_i$. Running the straight-line program $\Lambda = (a_1, a_2, \ldots, a_k)$ in G on input $(g_1, g_2, \ldots)$ means that we are successively substituting γ_i by f_{γ_i} and v_i by $h_i \in G$, where $h_i = f_{\gamma_i}(h_{i_1}, h_{i_2}, \ldots, h_{i_{r(\gamma_i)}})$, for $i = 1, 2, \ldots, k$. The output of the program is h_k. We use the notation $\Lambda(g_1, g_2, \ldots)$ for the output of running the program Λ on input $(g_1, g_2, \ldots)$. Of course, if $m = \max\{i : \alpha_i \text{ occurs in } \Lambda\}$, we can write $\Lambda(g_1, g_2, \ldots, g_m)$ with the same meaning.

In our data structure for solvable groups we have exactly what we need to set up the linear equations: the generators of the ith level correspond to the unit vectors in $GF(p)^{d_i}$, and for each $g \in G_{i-1}$ we can compute the vector corresponding to $gG_i \in G_{i-1}/G_i$. To set up the equations we proceed as follows. Let us denote the straight line program that represents $\omega \in \mathcal{R}$ by Λ_ω. For each $\omega \in \mathcal{R}$ we will compute a

$d_i|X|$ by d_i matrix D_ω and a vector b_ω. To get b_ω we compute $g = \Gamma_\omega(g_1, g_2, \ldots, g_{|X|})$ and then we compute the vector form of g^{-1}. To get the $(d_i(z-1)+k)$th column of D_ω ($1 \leq z \leq |X|$ and $1 \leq k \leq d_i$), we compute $h = \Gamma_\omega(h_1, h_2, \ldots, h_{|X|})$, where $h_m = g_m$ if $z \neq m$ and $h_z = g_z g_{ik}$ (using the notation from Chapter IX) and then we compute the vector representation of $g^{-1}h$. This way each relator yields d_i equations and finally we will have $|\mathcal{R}|d_i$ of them. We solve this system of equations to get the $|X|$ vectors, which should be lifted back to the permutation domain to get the modifiers for each g_z.

Extending the Constructive Presentation

When $p_i = p$, our task is to compute a new constructive presentation for H/G_{i+1}. We have a constructive presentation for H/G_i and we also have a constructive presentation for the vector space G_i/G_{i+1}, this latter is $\langle Y|\mathcal{T}\rangle$, where Y is a set of fresh free generators, and the relators are $\{[y_k, y_m] : 1 \leq k < m \leq j_{i+1}\} \cup \{y_m^p : 1 \leq m \leq j_{i+1}\}$. We define the extension of the map ϕ to $\overline{\phi}$ by $\overline{\phi}(y_k) = g_{ik}$. For $x \in X$, $\overline{\phi}(x) = \phi(x)$.

Luks in [Lu4] shows how to obtain a constructive presentation for H/N, given constructive presentations for H/K and for K/N, where $N < K$ are normal in H. We use this procedure to compute the new constructive presentation. This is done as follows.

Let H be a group and let $N < K$ be normal subgroups of H. Suppose we have a presentation $\langle X, \mathcal{R}\rangle$ of H/K realized via $\phi : X \to H$ and we also have a presentation $\langle Y, \mathcal{T}\rangle$ of K/N realized via $\psi : Y \to K$, where $X \cap Y = \emptyset$. We define $\overline{\phi} : X \cup Y \to H$ by $\overline{\phi}(x) = \phi(x)\ \forall x \in X$ and $\overline{\phi}(y) = \psi(y)\ \forall y \in Y$. We also define a set of new relations. For each $\omega \in \mathcal{R}$ find a word $\tau_\omega \in \mathcal{F}(Y)$ such that $\omega(\phi(X)) \equiv \tau_\omega(\psi(Y)) \bmod K$. Also, for each pair $(x, y) \in X \times Y$ find a word $\sigma_{x,y} \in \mathcal{F}(Y)$ such that $\psi(y)^{\phi(x)} \equiv$

$\sigma_{x,y}(\psi(Y))$ mod K. Let $\mathcal{U} = \mathcal{T} \cup \{\omega(X)\tau_\omega(Y)^{-1} : \omega \in \mathcal{R}\} \cup \{y^x\sigma_{x,y} : (x,y) \in X \times Y\}$. We define $\mathcal{R} \odot \mathcal{T} = \mathcal{U}$.

Lemma 10.8

Using the notation of the previous paragraph, $\langle X \cup Y | \mathcal{R} \odot \mathcal{T}\rangle$ is a presentation for H/N realized via $\overline{\phi} : X \cup Y \to H$. □

To find a word $\tau_\omega \in \mathcal{F}(Y)$ such that $\omega(\phi(X)) \equiv \tau_\omega(\psi(Y))$ mod K, in the case we are using this in the Sylow p-group algorithm, K/N is the elementary abelian p-group G_{i-1}/G_i, we have $\beta_i : G_{i-1}/G_i \to GF(p)^{p^{d_i}}$, and in the constructive presentation of it $\psi(Y)$ is mapped onto the standard generating system of $GF(p)^{p^{d_i}}$. So if the coefficient-vector for $\beta_i(\omega(\phi(X)))$ in the same basis is $(c_1, \ldots, c_{d_i})$, then $\tau_\omega = y_1^{c_1} \cdots y_{d_i}^{c_{d_i}}$.

To find the words τ_ω and $\sigma_{x,y}$ we have to start with $\phi(\omega) \in G_{i-1}$ and $\psi(y)^{\phi(x)} \in G_{i-1}$, respectively, and sift them through the level G_{i-1}/G_i, and whenever we perform an operation, we add a step to the straight-line program which prescribes that operation (the operands are the sifted word and a strong generator).

Let us examine the presentation we get applying this method in the Sylow p-subgroup algorithm. In that case, K/N will be G_i/G_{i+1} in the data structure for some i, and it is isomorphic with $GF(p)^{d_i}$ $(p = p_i)$. Let $Y = \{y_1, \ldots, y_{d_i}\}$, let $\psi : Y \to G_i$ such that $\psi(y_j) = g_{ij}$. Then $\langle Y, \psi | \mathcal{T}\rangle$ is a constructive presentation for G_i/G_{i+1}, where $\mathcal{T} = \{y_j^p : 1 \le j \le d_i\} \cup \{[y_j, y_k] : 1 \le j < k \le d_i\}$.

Definition 10.4

Let $X = \{x_1, \ldots, x_m\}$ and let $\langle X, \phi | \mathcal{R}\rangle$ be a constructive presentation for H/N. We say that this presentation has the tail-presentation property, if for each $X_i = \{x_i, x_{i+1}, \ldots, x_m\}$, $\langle X_i, \psi|_{X_i} | \langle X_i\rangle \cap \mathcal{R}\rangle$ is a constructive presentation for $\langle \psi(X_i)\rangle N/N$. □

Lemma 10.9

The constructive presentation Π that appears in the algorithm in Figure 15 always has the tail presentation property.

Proof

It is trivial that the constructive presentation described above for G_i/G_{i+1} has the property and it is easy to see that if both $\langle\{x_1,\ldots,x_j\},\phi|\mathcal{R}\rangle$ and $\langle x_{j+1},\ldots,x_k\},\psi|\mathcal{T}\rangle$ have the tail-presentation property then so does $\langle\{x_1,\ldots,x_k\},\phi\cup\psi|\mathcal{R}\odot\mathcal{T}\rangle$. From this, by induction on the length of the construction, we get the statement of the lemma. □

This means that we can apply Lemma 10.6 to set up equations for one generator at a time when we compute the complement, starting with the last one. There is always a complement containing the already computed part, since any p-subgroup of a group is contained in some Sylow p-subgroup, so by Lemma 10.6 there is always a solution for the system of linear equations.

Complexity of the Sylow p-Subgroup Algorithm

Let l be the length of the composition series of G, that is $l = d_1 + \cdots + d_m$. Then we will have to set up and solve $O(l)$ systems of equations. To compute a complement for G_{i-1}/G_i in H/G_i when the dimension of G_{i-1}/G_i is d_i, we solve a series of systems of equations, each for d_i unknowns, with a total of $O(l^2d_i)$ equations (this is the number of relations). To set up these equations, we have to evaluate $O(l^2d_i)$ straight-line programs, each requiring $O(l)$ group operations and one permutation-vector transformation. To solve the system of equations costs $O(l^2d_i^\alpha)$ field operations. The asymptotically best known algorithms for doing linear algebra provide $\alpha < 2.4$

(see e.g. [CLR]), but if we use good old Gaussian elimination, we can get $\alpha = 3$. To get the total number of field operations during the computation we have to sum this over the i's for which G_{i-1}/G_i is not a p-group, giving a total of $O(l^{2+\alpha})$ field operations, and $O(l^4)$ permutation operations. Since $l = O(n)$, where n is the size of the permutation domain, the whole running time is $O(n^5)$. These results seem to match those in [EW], where similar tasks are performed in a slightly different setting. If we haven't used the result of Lemma 10.9, then we should have had to solve linear equations of size $O(ld_i)$ unknowns and $O(l^2 d_i)$, giving a total cost of $O(l^{2+2\alpha})$.

Experiments

We conducted experiments comparing the techniques presented in this chapter with built-in methods of GAP [Sch]. In the experiments we computed Sylow p-subgroups of several solvable permutation groups using two methods. The first method involves built-in GAP functions, first to convert the permutation group into an Ag-group (GAP's term for solvable groups represented by a power-commutator presentation), then to convert it into a so-called Special Ag-group, in which getting the Sylow subgroups is a trivial matter, finally mapping the result back to the permutation domain. The second method is based on the algorithm discussed in this chapter and on the data structure presented in Chapter IX. GAP also has a method for finding Sylow p-subgroups of permutation groups, but that is incomparably slower than any of the two methods described above.

The groups in the experiment were the following:

G_1 a solvable linear group of size $2^7 \cdot 3^9 \cdot 7$ in a primitive permutation representation on $3^8 = 6561$ points,

G_2 a supergroup of G_1 on the same points, of size $2^{10} \cdot 3^9 \cdot 7$.

G_3 a linear group of size $2^7 \cdot 3^5$ represented on $3^4 = 81$ points.

(The three groups above were suggested by W.M. Kantor)

$G_4 = S_3 \wr S_4$, size $2^7 \cdot 3^5$, on 12 points.

$G_5 = S_4 \wr S_4$, size $2^{15} \cdot 3^5$, on 16 points.

$G_6 = G_5 \times Syl_3(S_{21}) \times G_4$, size $2^{22} \cdot 3^{19}$, on 49 points.

$G_7 = G_6 \times G_6$, size $2^{44} \cdot 3^{38}$, on 98 points.

$G_8 = G_7 \times G_6$, size $2^{66} \cdot 3^{57}$, on 147 points.

$G_9 = G_8 \times G_6$, size $2^{88} \cdot 3^{76}$, on 196 points.

$G_{10} = G_9 \times G_6$, size $2^{110} \cdot 3^{95}$, on 245 points.

$G_{11} = G_1 \wr S_3$, size $2^{22} \cdot 3^{28} \cdot 7^3$, on $3^9 = 19683$ points.

$G_{12} = G_2 \wr S_4$, size $2^{43} \cdot 3^{37} \cdot 7^4$, on $3^8 \cdot 2^2 = 26244$ points.

$G_{13} = G_3 \wr S_4$, size $2^{31} \cdot 3^{21}$, on $3^4 \cdot 2^2 = 324$ points.

The groups were given by "nice" generators, that is, ones that reflect the product structure. Then a set of subgroups of each group was computed by choosing four random elements as generators - in the majority of the cases these random elements generated the whole group (see the second column – composition length – of Table 1 and Table 2).

The experiments showed that although both methods are sensitive to the "niceness" of generators, the one composed from the built-in functions takes considerably

(2 to 35 times) more time to finish even on smaller groups if the generators are not "nice". Our method performed within a time factor of two, and if the size of the "random" subgroup was smaller than the original group, then the times even went down.

The two methods finish in time within a factor of two on small groups, the built-in method is favorable (by a factor of 2 to 4) on the primitive groups G_1 and G_2 (relatively large permutation domain compared to the size of the group). In the series G_6 to G_{10} we tried to get an empirical growth rate comparison between the two methods, as far as the computations could be done in reasonable time. The timing results suggest that the built-in method has a higher growth rate than our method. This is especially true for the random generators case, when the relators in the Ag-presentation are likely to contain more generators, so the multiplication of two elements (by the method called collection) takes much more time.

In the experiments we measured the time spent in each of the steps for both methods. For the built-in method these parts are computing the point-stabilizer series for the permutation group, then converting the permutation group into an Ag-group, then converting the Ag-group into a special Ag-group. The time required for finding a Sylow subgroup of the special Ag-group and converting it back to a permutation group is negligible compared to the rest. For our method the sub-tasks are computing the linear stucture, then computing the strong generating sequence for the elementary abelian structure (at which point we have about the same information about the structure of the group as we have with an Ag-representation) then finally computing the Sylow p-subgroup.

As expected, the bottleneck in the computations for small sizes is the middle part, that is, computing the strong generating set, but as the sizes grow, the bulk of the computation shifts to the last one (computing the Sylow subgroup).

Table 1: Nice generators, whole groups

	comp len	struct	sgs	Syl	pss	Ag	SAg	total perm	total ag
G_1	17	8.4	5.7	0.1	2.7	2.2	0.3	14.3	5.2
G_2	20	21.5	6.7	0.1	3.2	3.8	0.4	28.3	7.5
G_3	12	0.19	0.07	0.01	0.06	0.05	0.16	0.27	0.27
G_4	12	0.03	0.05	0.01	0.05	0.05	0.15	0.09	0.25
G_5	20	0.05	0.09	0.11	0.1	0.2	0.4	0.25	0.7
G_6	41	0.1	0.3	1.1	0.4	2.7	1.4	1.5	4.4
G_7	82	0.1	2.0	12.6	1.3	40.1	5.3	14.8	46.7
G_8	123	0.3	2.9	70.2	4.7	205.3	14.1	73.3	224.1
G_9	164	0.4	6.5	281.5	9.7	668.0	29.3	288.3	707.0
G_{10}	205	0.5	10.4	1144.4	17.9	1675.8	54.2	1155.3	1747.9
G_{11}	53	21.1	112.0	42.0	20.7	225.4	3.0	176.0	249.2
G_{12}	84	59.1	465.7	192.1	82.6	1380.1	12.1	716.9	1474.8
G_{13}	52	0.4	2.2	1.0	0.6	6.3	4.9	3.6	11.9

The execution times are tabulated in Table 1 and Table 2. The numbers represent seconds of processor time as reported by the GAP function Runtime(), on a Sun Ultrasparc-1 machine, with 64 Megabytes of memory workspace for GAP.

Table 2: Subgroups generated by random elements

	comp len	struct	sgs	Syl	pss	Ag	SAg	total perm	total ag
H_1	17	7.8	17.2	0.2	2.5	2.6	0.5	25.1	5.6
H_2	20	10.0	45.8	0.9	2.9	5.9	0.8	56.7	9.5
H_3	12	0.19	0.21	0.00	0.06	0.04	0.23	0.40	0.33
H_4	12	0.02	0.07	0.01	0.06	0.05	0.29	0.1	0.4
H_5	20	0.03	0.2	0.1	0.1	0.2	0.9	0.3	1.2
H_6	37	0.1	0.8	2.9	0.4	2.0	6.1	3.9	8.5
H_7	71	0.2	4.4	8.9	2.0	22.3	39.4	13.5	63.7
H_8	102	0.2	18.3	39.2	3.9	97.9	369.2	57.8	471.0
H_9	128	0.3	36.9	107.2	6.8	259.2	2948.9	144.4	3218.2
H_{10}	164	2.2	99.2	446.4	16.1	748.0	20665.7	547.8	21475.9
H_{11}	53	28.7	702.4	68.9	63.3	922.8	42.4	800.0	1028.5
H_{12}	84	85.9	10945.8	454.2	177.0	*	*	11485.9	*
H_{13}	52	0.3	16.1	1.5	0.8	10.7	48.2	17.9	59.7

BIBLIOGRAPHY

[AHU] A. V. Aho, J. E. Hopcroft, J. D. Ullman *The Design and Analysis of Computer Algorithms* Addison-Wesley, Reading, Mass., 1974.

[At] M. D. Atkinson, *An algorithm for finding the blocks of a permutation group,* Math. Comp., 29 (1975), pp. 911–913.

[Ba] L. Babai, *On the Length of Subgroup Chains in the Symmetric Goup*, Comm. in Alg., 14 (1986), pp. 1729–1736.

[BCFS] L. Babai, G. Cooperman, L. Finkelstein, Á. Seress, *Nearly Linear Time Algorithms for Permutation Groups with a Small Base*, Proc. 1991 ACM-SIGSAM Inter. Symp. on Symbolic and Algebraic Comp., 1991, pp. 200-209.

[BKL] L. Babai, W.M. Kantor, E.M. Luks, *Computational Complexity and the Classification of Finite Simple Groups*, Proc. 24^{th} IEEE FOCS (1983), pp. 161–168.

[BLS1] L. Babai, E. M. Luks, Á. Seress, *Fast Management of Permutation Groups I*, Technical Report CIS-TR-94-22, U. of Oregon, October 1994.

[BLS2] L. Babai, E. M. Luks, Á. Seress, *Fast Management of Permutation Groups II*, In preparation

[Be] R. Beals *Computing Blocks of Imprimitivity for Small-Base Groups in Nearly Linear Time* in Groups and Computation, DIMACS Series in Discrete Mathematics and Theoretical Computer Science, 11, Amer. Math. Soc, 1993, ed. L. Finkelstein and W.M. Kantor, pp. 17–26.

[Bu] G. Butler, *An Analysis of Atkinson's Algorithm* Technical Report 259, Basser Dept. of Comp. Sci., U. of Sydney, 1985.

[Ca] P. J. Cameron, *Finite Permutation Groups and Finite Simple Groups*, Bull. London Math. Soc., 13 (1981), pp. 1–22.

[CNW] F. Celler, J. Neubüser, C. R. B. Wright, *Some Remarks on the Computation of Complements and Normalizers in Soluble Groups* Acta Applicandae Mathematicae 21 (1990), pp. 57–76.

[CLR] T. H. Cormen, C. E. Leiserson, R. L. Rivest, *Introduction to Algorithms* The MIT Press/McGraw-Hill, 1990.

[CP] J. Cannon and C. Playoust, *An Introduction to MAGMA*, School of Mathematics and Statistics, U. of Sydney, 1993.

[Di] J. D. Dixon, *The Solvable Length of a Solvable Linear Group*, Math. Zeitschr. 107 (1968), pp. 151–158.

[DM] J. D. Dixon, B. Mortimer, *Permutation Groups*, Graduate Texts in Mathematics, vol. 163. Springer, New York, 1996.

[EW] B. Eick, C. R. B. Wright, *Computing Subgroups of Finite Solvable Groups* in preparation

[FHL] M. L. Furst, J. Hopcroft, E. M. Luks, *Polynomial-Time Algorithms for Permutation Groups*, Proc 21st IEEE FOCS, 1980, pp. 36–41.

[GHLSW] Z. Galil, C. M. Hoffmann, E. M. Luks, C. P. Schnorr, A. Weber *An $O(n^3 \log n)$ Deterministic and an $O(n^3)$ Las Vegas Isomorphism Test for Trivalent Graphs*, Journal of the ACM, Vol 34, No. 3, July 1987, pp. 513-531.

[Ha] M. Hall *Theory of groups*, Chelsea, New York, 1959.

[Hu] B. Huppert, *Endliche Gruppen I* Springer-Verlag, Berlin, 1967.

[Je] M. A. Jerrum, *A Compact Representation for Permutation Groups*, Proc 23rd IEEE FOCS (1982) pp. 126–133.

[Ka] W. M. Kantor, *Sylow's Theorem in Polynomial Time*, J. Comp. Sys. Sci. 30 (1985), pp. 359–394.

[KL] W. M. Kantor, E. M. Luks *Computing in Quotient Groups*, Technical Report CIS-TR-90-07, U. of Oregon, March, 1990.

[KLM] W. M. Kantor, E. M. Luks and P. D. Mark *Sylow Subgroups in Parallel* (in preparation)

[KT] W. M. Kantor, D. E. Taylor, *Polynomial-time Versions of Sylow's Theorem*, Journal of Algorithms 9 (1988) pp. 1–17.

[Kn] D. E. Knuth *Efficient Representation of Perm Groups*, Combinatorica 11 (1991), pp. 33–44.

[Lu1] E. M. Luks, *Isomorphism of Graphs of Bounded Valence Can Be Tested in Polynomial Time*, Journal of Computer and System Sciences, Vol. 25, No. 1, August 1982.

[Lu2] E. M. Luks, *Lectures on Polynomial-time Computations in Groups*, Technical Report CIS-TR-90-21, U. of Oregon, December 1990.

[Lu3] E. M. Luks, *Permutation Groups and Polynomial-time Computation*, Groups and Computation, DIMACS Series in Discrete Mathematics and Theoretical Computer science, Vol. 11, pp. 139–176.

[Lu4] E. M. Luks *Computing in solvable matrix groups* Proceedings of the 33rd IEEE Symposium on the Foundations of Computer Science, 1992, pp. 111-120.

[LRW1] E. M. Luks, F. Rákóczi and C. R. B. Wright, *Computing Normalizersers in Permutation p-groups*, Proceedings of the International Symposium on Symbolic and Algebraic Computation (ISSAC), Oxford, July 1994, pp. 139–146.

[LRW2] E. M. Luks, F. Rákóczi and C. R. B. Wright, *Some Algorithms for Nilpotent Permutation Groups*, J. Symbolic Computation 23 (1997), 335–354.

[Mc] P. McKenzie, *Parallel Complexity and Permutation Groups*, Doctoral Thesis, Technical Report 173/84, Dept. of Computer Science, University of Toronto, 1984.

[Mi1] G.L. Miller, *Graph Isomorphism, General Remarks*, J. Comp. Sys. Sci. 18 (1979), pp. 128–142.

[Mi2] G.L. Miller, *Isomorphism of k-contractible Graphs, a Generalization of Bounded Valence and Bounded Genus*, Inform. and Control 56 (1983), pp. 1-20.

[Ma] P. D. Mark, *Parallel Computation of Sylow Subgroups in Solvable Groups* Groups and Computation, DIMACS Series in Discrete Mathematics and Theoretical Computer science, Vol. 11, pp. 177–188.

[Neu] J. Neubüser, *An Invitation to Computational Group Theory*, in preparation

[Pá] P. P. Pálfy, *A Polynomial Bound for the Orders of Primitive Solvable Groups*, J. Algebra, 77 (1982), pp. 127–137

[Rá] F. Rákóczi, *Fast Recognition of Nilpotent Permutation Groups*, Proceedings of the International Symposium on Symbolic and Algebraic Computation (ISSAC), Montreal, July 1995, pp. 265-269.

[RC] R.C. Read and D.G. Corneil, *The Graph Isomorphism Disease*, J. Graph Theory, 3 (1979), pp. 339-363.

[SchSe] M. Schönert and Á. Seress *Finding blocks of imprimitivity in small-base groups in nearly linear time* Proceedings of th International Symposium on Symbolic and Algebraic Computation (ISSAC), Oxford, July 1994, pp. 154–157.

[Sch] M. Schönert et al., GAP, Groups, Algorithms and Programming, Lehrstuhl D für Mathematik, Rheinisch Westfälische Technische Hochschule, Aachen, Germany, 3rd edition, 1993.

[Se] Á. Seress, *Nearly Linear Time Algorithms for Permutation Groups: an Interplay between Theory and Practice*, Acta Applicandae Mathematicae, to appear

[Si1] C. C. Sims, *Computational methods in the study of permutation groups*, Computational Problems in Abstract Algebra, Oxford 1967. Pergamon Press, Oxford, 1970, pp. 169–183.

[Si2] C. C. Sims, *Computing the Order of a Solvable Permutation Group* J. Symbolic Computation 9 (1990), pp. 699–705.

[Tar] R. E. Tarjan *Efficiency of a good but not Linear set union algorithm.* J. ACM 22, 2, Apr 1975, pp. 215-225.

[Wie] H. Wielandt *Finite permutation groups*, Academic Press, 1964.

[Wo] T. R. Wolf, *Solvable and Nilpotent Subgroups of* $GL(n, q^m)$, Canad. J. Math. 34 (1982), pp. 1097–1111.