Low-complexity computations for nilpotent subgroup problems

Jeremy Macdonald,* Alexei Miasnikov† Denis Ovchinnikov‡

July 5, 2017

Abstract
We solve the following algorithmic problems using $TC^0$ circuits, or in logspace and quasilinear time, uniformly in the class of nilpotent groups with bounded nilpotency class and rank: subgroup conjugacy, computing the normalizer and isolator of a subgroup, coset intersection, and computing the torsion subgroup. Additionally, if any input words are provided in compressed form as straight-line programs or in Mal’cev coordinates the algorithms run in quartic time.

**Contents**

1 Introduction 1
2 Background 3
   2.1 Nilpotent presentations 3
   2.2 Subgroups 4
   2.3 Logspace, $TC^0$, and compressed words 4
   2.4 Fundamental algorithms for nilpotent groups 7
3 Algorithmic problems 9
   3.1 Subgroup conjugacy and normalizers 10
   3.2 Coset intersection 13
   3.3 Torsion subgroup 19
   3.4 Isolator 21

1 Introduction

This is the third paper in a series on complexity of algorithmic problems in finitely generated nilpotent groups. In the first paper [MMNV15], we showed

*Concordia University, jeremy.macdonald@concordia.ca
†Stevens Institute of Technology, Alexei.Miasnikov@stevens.edu
‡Stevens Institute of Technology, dovchinn@stevens.edu
that the basic algorithmic problems (normal forms, conjugacy of elements, subgroup membership, centralizers, presentation of subgroups, etc.) can be solved by algorithms running in logarithmic space and quasilinear time. Further, if the problems are considered in ‘compressed’ form with each input word provided as a straight-line program, we showed that the problems are solvable in polynomial time. The second paper [MW17] pushed the complexity of these problems lower, showing that they may be solved by TC$^0$ circuits. Here we expand the list of algorithmic problems for nilpotent groups which may be solved in these low complexity conditions to include several fundamental problems concerning subgroups.

Note that in group theory algorithmic problems for subgroups of groups are usually much harder then the basic algorithmic problems mentioned above. Nevertheless, we present here algorithms for deciding the conjugacy of two subgroups of a finitely generated nilpotent group $G$, finding the normalizer and the isolator of a given subgroup of $G$, finding the torsion subgroup $T(G)$ of $G$, and finding the intersection of two cosets of subgroups of $G$, all of which may be implemented by TC$^0$ circuits, or run in logarithmic space and quasilinear time on a (multi-tape) Turing machine. Furthermore, the compressed versions of these problems are solvable in polynomial (specifically, quartic) time. All of the algorithms work uniformly over finitely generated nilpotent groups (i.e. the group may be included in the algorithm’s input), however the complexity bounds depend on the nilpotency class and the rank (number of generators) of the presentation. When both are bounded, we solve all the problems uniformly in TC$^0$ or logspace and quasilinear time.

Algorithmic problems in nilpotent groups have been studied for a long time. On the one hand, it was shown that many of them are decidable and many sophisticated decision algorithms were designed (see, for example, the pioneering paper [KRR+69] by Kargapolov et al. published in 1969 and the books [Sim94] and [HEO05] for more recent techniques); on the other hand, there are some which have been known to be undecidable for some time (for instance, decidability of equations [Rom77]). Recent work by a variety of authors has introduced a host of decidable/undecidable problems. New undecidable problems, including the knapsack problem, commutator and retract problems are described in [Loh15], [KLZ15], [MT16], and [Rom16], while positive decidability results for direct product decompositions and equations in the Heisenberg group are described in [BMO16] and [DLS15]. Decidability and undecidability results for equations over random nilpotent groups are also given in [GMO16a] and [GMO16b].

However, it seems that this paper together with [MMNV15] and [MW17] present the first thorough attempt to study the complexity of the problems, beyond the decidable/undecidable dichotomy. In fact, it seems this is currently the only known large class of non-abelian groups where the major algorithmic problems are shown to have low space and time complexity. Another large class of such groups is, perhaps, the class of finitely generated free groups given by the standard presentations. Even there, if the free groups are given by arbitrary finite presentations the complexity of the algorithmic problems is still mostly
unknown.
We have not yet mentioned one of the fundamental algorithmic problems in nilpotent groups: the isomorphism problem. It is decidable due to the famous result of Grunewald and Segal [GS80]. Nevertheless, not much is known about its complexity.

**Problem 1.** Is the isomorphism problem in finitely generated nilpotent groups decidable in polynomial time? Exponential time?

2 Background

This section describes, summarizing from [MMNV15] and [MW17], how we will represent finitely generated nilpotent groups (§2.1) and their subgroups (§2.2), and gives black-box descriptions of several algorithms that we will be using as subroutines (§2.4). We also give a brief introduction to the $\text{TC}^0$ circuit model of computation, logspace computations, and the use of compressed words in algorithmic problems over groups (§2.3).

2.1 Nilpotent presentations

Let $G$ be a finitely generated nilpotent group of nilpotency class $c$. Then $G$ has lower central series

$$G = \Gamma_1 \triangleright \Gamma_2 \triangleright \ldots \triangleright \Gamma_c \triangleright \Gamma_{c+1} = 1$$

with $\Gamma_{i+1} = [G, \Gamma_i]$ for $i > 1$. From this series we derive a presentation for $G$, as follows.

Each $\Gamma_i/\Gamma_{i+1}$ is a finitely generated abelian group. We select and fix a finite generating set $a_{s_i-1+1}\Gamma_{i+1}, \ldots, a_{s_i}\Gamma_{i+1}$ for $\Gamma_i/\Gamma_{i+1}$ and put

$$A = \{a_1, a_2, \ldots, a_m\}.$$ 

For each $j = 1, \ldots, m$, if $s_{i-1} + 1 \leq j \leq s_i$, we denote by $e_j$ the order of $a_j\Gamma_{i+1}$ in $\Gamma_i/\Gamma_{i+1}$, using $e_j = \infty$ when the order is infinite. Denote

$$T = \{i \mid e_i < \infty\}.$$ 

Provided that each generating set above is chosen to correspond to a primary or invariant factor decomposition of $\Gamma_i/\Gamma_{i+1}$, every element $g \in G$ may be written uniquely in *Mal’cev normal form* as

$$g = a_1^{\alpha_1}a_2^{\alpha_2}\cdots a_m^{\alpha_m}$$ (1)

where $\alpha_i \in \mathbb{Z}$ and if $i \in T$ then $0 \leq \alpha_i < e_i$. The set $A$ is called a *Mal’cev basis* of $G$ and the integers $(\alpha_1, \ldots, \alpha_m)$ are the *Mal’cev coordinates* of $g$.

For each $i = 1, \ldots, m$, denote $G_i = \langle a_i, \ldots, a_m \rangle$. An essential fact, which follows from the definition of the lower central series, is that for any $i < j$,

$$[a_i, a_j] \in G_\ell$$
for some $\ell > j$. From this it follows that relations of the form

$$a_ja_i = a_ia_j^{\alpha_{ij}}a_{\ell+1}^{\alpha_{ij}(\ell+1)} \cdots a_m^{\alpha_{ij}m}$$

(2)

$$a_j^{-1}a_i = a_ia_j^{-1}a_{\ell+1}^{\beta_{ij}(\ell+1)} \cdots a_m^{\beta_{ij}m},$$

(3)

with $\ell > j$, hold in $G$. In addition, for each $i \in T$ there is a relation of the form

$$a_i^{e_i} = a_{\ell+1}^{\mu_i} \cdots a_m^{\mu_m}$$

(4)

where $\ell > j$. The set $\{a_1, \ldots, a_m\}$, viewed as an abstract set of symbols, together with relations (2)--(4) then form a presentation for $G$ called a nilpotent presentation. In fact, any presentation of this form defines a nilpotent group. Such a presentation is called consistent if the order of each $a_i$ modulo $\langle a_{i+1}, \ldots, a_m \rangle$ is precisely $e_i$. Note that $e_i = 1$ is permitted in a nilpotent presentation.

For low-complexity algorithms, an essential property of nilpotent presentations is the following (see [MMNV15] Thm. 2.3 and Lem. 2.5): if $w$ is any word over $A^\pm$, then the length of the Mal’cev normal form (1) of the element $g$ corresponding to $w$ in $G$ is bounded by a polynomial function of the length of $w$, with the degree of the polynomial depending on the nilpotency class $c$ and number of generators $r$ of $G$. This fact plays a crucial role in solving efficiently the fundamental algorithmic problems in finitely generated nilpotent groups.

### 2.2 Subgroups

All of our results concern subgroups of finitely generated nilpotent groups. For every subgroup $H \leq G$ (all of which are, necessarily, finitely generated), one may define a unique generating set $(h_1, \ldots, h_s)$ called the full-form sequence for $H$. The precise definition was given in [Sim94] (and is reviewed in [MMNV15]), but we mention here only three facts about $(h_1, \ldots, h_s)$ that we will need.

First, let $B$ be the matrix in which row $i$ is the row vector consisting of the Mal’cev coordinates of $h_i$. Then $B$ is in row echelon form and does not contain zero rows. We denote by $\pi_i$ the pivot column of row $i$ of $B$. Since this column corresponds to generator $a_{\pi_i}$, the Mal’cev normal form of $h_i$ begins with $a_{\pi_i}$, so $h_i \in G_{\pi_i} = \langle a_{\pi_i}, \ldots, a_m \rangle$.

Second, the number of generators $s$ is bounded by the length $m$ of the Mal’cev basis. Third, every element $h \in H$ can be uniquely presented in the form

$$h = h_1^{\beta_1} \cdots h_s^{\beta_s},$$

where $\beta_i \in \mathbb{Z}$ and $0 \leq \beta_i < e_{\pi_i}$ if $\pi_i \in T$. Hence

$$H = \{h_1^{\beta_1} \cdots h_s^{\beta_s} | \beta_i \in \mathbb{Z} \text{ and } 0 \leq \beta_i < e_{\pi_i} \text{ if } \pi_i \in T \}.$$
The set $A$ consists of a set of symbols, say $x_1, \ldots, x_n$, which denote group generators, and a few extra symbols used to separate different parts of the input (commas to separate relators etc.). We will be computing $f(x)$ using logarithmic space or using $\text{TC}^0$ circuits. We recall both of these notions below.

**Logspace.** A $c$-logspace transducer, where $c > 0$ is a constant, is a multi-tape Turing machine consisting of the following tapes: an ‘input’ tape which is read-only, a constant number of read-write ‘work’ tapes, and a write-only ‘output’ tape. For any input of length $L$, which is provided on the input tape, the amount of space the transducer is allowed to use on each work tape is $c \log(L)$. The output of the machine is the content of the output tape. A function $f$ is said to be logspace computable, or more casually the associated problem is solvable in logarithmic space, if there exists a constant $c$ and a $c$-logspace transducer that produces $f(x)$ on the output tape for any input $x$ appearing on the input tape.

Though computation on a $c$-logspace transducer puts a bound only on space resources, a polynomial time bound of $O(L^c)$ is forced by the fact that the machine may not enter the same configuration twice (otherwise it will loop infinitely) and the number of configurations is bounded by a polynomial function of the input length. The degree $c$ may be very high, and for this reason it is also desirable to show directly that our algorithms run in low-degree polynomial time, in particular quasilinear time (i.e. $O(L \log^k(L))$ for some constant $k$).

Most of our algorithms invoke other logspace algorithms as subroutines, and as such we need to compute compositions $f \circ g$ of logspace computable functions. A standard argument shows that $f \circ g$ is again logspace computable, but in computing $(f \circ g)(x)$ in this way, each symbol of $g(x)$ is recomputed each time it is needed in computation of $f$, which may give a significant increase in time complexity. However, if the output $g(x)$ is always of size $O(\log(L))$, one may simply compute $g(x)$ first, store the output on the work tape, and then proceed to compute $f(g(x))$. This is the case in all of our algorithms, so in this case the time complexity of $g$ is simply added to the overall time complexity.

**$\text{TC}^0$ circuits.** A $\text{TC}^0$ circuit with $n$ inputs is a boolean circuit of constant depth using NOT gates and unbounded fan-in AND, OR, and MAJORITY gates, such that the total number of gates is bounded by a polynomial function of $n$. A MAJORITY gate outputs 1 when more than half of its inputs are 1. A function $f(x)$ is $\text{TC}^0$-computable (more casually, ‘an algorithm is in $\text{TC}^0$’) if for each $n$ there is a $\text{TC}^0$ circuit $F_n$ with $n$ inputs which produces $f(x)$ on every input $x$ of length $n$. Essential for our purposes is the fact that the composition of two $\text{TC}^0$-computable functions is again $\text{TC}^0$-computable.

Since this definition of being computable only asserts that such a family $\{F_n\}_{n=1}^\infty$ of circuits exists, one normally imposes in addition a uniformity condition stating that each $F_n$ is constructible in some sense. We will only be concerned here with standard notion of DLOGTIME uniformity, which asserts that there is a random-access Turing machine which decides in logarithmic time
whether in circuit $F_n$ the output of gate number $i$ is connected to the input of gate $j$, and determines the types of gates $i$ and $j$. We refer the reader to [Vol99] for further details on $TC^0$.

To put our results in context, we remind the reader of the following inclusions of complexity classes:

$$TC^0 \subseteq \text{LOGSPACE} \subseteq P \subseteq \text{NP}.$$  

It is not known whether any of these inclusions is strict. Though every $TC^0$-computable function is also logspace-computable and polynomial-time computable, our algorithm descriptions also give direct proofs of membership in these classes.

**Compressed words.** We are also interested in algorithms that run efficiently when the input is given in compressed format. The use of Mal’cev coordinates provides a natural compression scheme for elements of $G$: each $g \in G$ may be encoded by a tuple of integers (its Mal’cev coordinates) written in binary. Notice that if the size $m$ of the Mal’cev basis is bounded, a normal form of length $n$ may be encoded by $O(\log n)$ bits. Since every finitely generated nilpotent group has a Mal’cev basis, it is natural to consider algorithmic problems in which input words represented in this compact way. Of course, such ‘compressed problems’ are, in terms of computational complexity, more difficult than their uncompressed siblings.

Since we will consider uniform algorithms, in which a finitely generated nilpotent group $G$ is given by an arbitrary presentation as part of the input, we also consider two other compression schemes which do not depend on a the specification of a Mal’cev basis. First, we may simply allow exponents to be encoded in binary. In this scheme, a word is encoded as a product of tuples $(g, m)$, representing $g^m$, where $g$ is a group generator or, recursively, a word of this form, and $m$ is a binary integer. For example, $(x^4 y^2) x^{-6}$ is encoded as $(((x, 0100)(y, 0010)), 1000)(x^{-1}, 0110)$.

Second, we consider straight-line programs, that is, context-free grammars that generate exactly one string. Formally, a straight-line program or compressed word over an alphabet $A$ consists of a set $\{A_1, A_2, \ldots, A_n\}$ called the non-terminal symbols and for each non-terminal symbol $A_i$ a production rule either of the form $A_i \rightarrow A_j A_k$ with $j, k < i$, or of the form $A_i \rightarrow a$ where $a \in A \cup \{\epsilon\}$ with $\epsilon$ denoting the empty word. The non-terminal $A_n$ is termed the root, and one ‘expands’ the compressed word by starting with the one-character word $A_n$ and successively replacing any non-terminal with the right side of its production rule until only symbols from $A$ remain. The number $n$ of non-terminal symbols is the size of the program. Compression arises since a program of size $n$ may expand to a word of length $2^n - 1$. We refer the reader to the survey article [Loh12] and the monograph [Loh14] for further information on compressed words, or to the introduction of [MMNV15] for some brief remarks.
2.4 Fundamental algorithms for nilpotent groups

Throughout this paper, we make extensive use of algorithms described in [MMNV15] and [MW17]. We give below a summary of some of the most heavily-used ones, and we will use the names listed here, in **bold text**, to refer to their use.

- **Full-form Sequence**: Given $H \leq G$, compute the full-form generating sequence for $H$.

- **Membership**: Given $g \in G$ and $H \leq G$, determine if $g \in H$ and if so, compute the unique expression $g = h_1^{a_1} \cdots h_s^{a_s}$ where $(h_1, \ldots, h_s)$ is the full-form sequence for $H$.

- **Subgroup Presentation**: Given $H \leq G$, compute a consistent nilpotent presentation for $H$.

- **Conjugacy**: Given $g, h \in G$, produce $x \in G$ such that $g^x = h$ or determine that no such $x$ exists.

- **Centralizer**: Given $g \in G$, compute a generating set for the centralizer of $g$ in $G$.

- **Kernel**: Given $K \leq G$ and $\phi : K \to G_1$, produce a generating set for the kernel of $\phi$.

- **Preimage**: Given $K \leq G$, $\phi : K \to G_1$, and $h \in G_1$ guaranteed to be in $\phi(K)$, produce $k \in K$ such that $\phi(k) = h$.

We will need some further details regarding the input/output of these algorithms as well as their complexity.

**Input.** In each algorithm, we fix in advance two integers $c$ and $r$. The ambient nilpotent groups $G$ and $G_1$ are part of the input (thus the algorithms are ‘uniform’) but must be of nilpotency class at most $c$ and be presented using at most $r$ generators for the complexity bounds given below to be valid. Group elements are given as words over the generating set(s), subgroups are specified by finite generating sets, and $\phi$ is given by listing the elements $\phi(k)$ for each given generator $k$ of $K$. The length $L$ of the input is the sum of the lengths of all relators in $G$ and $G_1$ plus the lengths of all input words.

**Output.** Each output word is given as a word over the original generating set except possibly in **Full-form sequence** and **Membership**. In these cases, the algorithm converts to a nilpotent presentation of $G$, if one is not already provided, and provides the output words in the new generators (the isomorphism may also be provided, see Lemma 3 below). In **Centralizer** and **Kernel**, if the original presentation of $G$ is already a nilpotent presentation, one may assume that the subgroup generating set in the output is the full-form sequence.

In every case, the total length of each output word is bounded by a polynomial function of $L$ and the number of output words is bounded by a constant. Optionally, the output words may be given by their Mal’cev coordinates.
**Complexity.** Each algorithm may be implemented on a logspace transducer, and if so runs in time quasilinear in $L$. The proofs are given in [MMNV15]. Alternatively, each problem may be solved using TC$^0$ circuits, as proved in [MW17].

**Compressed inputs.** Each algorithm may also be run ‘with compressed inputs’. In this case, any input word (including group relators) may be provided by (binary) Mal’cev coordinates, words with binary exponents, or straight-line programs, as described in §2.3. We will measure the size of the input in terms of the number $n$ of input words and the maximum size $M$ of any single input word (in number of bits or number of non-terminal symbols). The space complexity of each algorithm is then $O(M)$ (it does not depend on $n$) and the time complexity is $O(nM^3)$. All output is provided in the corresponding compressed format. Although each input word, in its expanded form, may have length $O(2^M)$, the polynomial bound for the length of output words implies that each output word, in expanded form, has length $O(2^{dM})$ where $d$ is the degree of the aforementioned polynomial bound. Since $d$ is constant, the compressed size of each output remains $O(\log(2^{dM})) = O(M)$.

**Remark 2.** We place no restriction on the number $n$ of input words. In all of the algorithms, any variable-sized set of input words (e.g. list of subgroup generators, group relators) will be fed as input to the matrix reduction algorithm described in Thm. 3.4 of [MMNV15] and processed in the ‘piecewise’ manner described there, one word at a time. After this, sets of words usually only appear as full-form sequences for subgroups, the number of which is always bounded by a constant. The value $n$ contributes a linear factor to the time complexity of this algorithm (in both uncompressed and compressed cases), but does not contribute to the space complexity.

While neither these algorithms nor the ones we describe in this paper require that the input groups $G$ and $G_1$ be given by a nilpotent presentation, this form is used internally by all of the algorithms. Converting to such a presentation is accomplished as follows.

**Lemma 3.** Let $c$ and $r$ be fixed integers. There is an algorithm that, given a finitely presented nilpotent group $G = \langle X \mid R \rangle$ of nilpotency class at most $c$ and with $|X| \leq r$, a finite set $Y \subset G$, and a word $h$ over $X^\pm$ guaranteed to be in the subgroup $H = \langle Y \rangle$, produces

- a consistent nilpotent presentation $\langle Y' \mid S \rangle$ for $H$, in which binary numbers are used to encode exponents in the relators $S$,
- a map $\phi : Y' \to (Y^\pm 1)^*$ which extends to an isomorphism $\langle Y' \mid S \rangle \to H$, and
- a binary integer tuple $h'$ giving the Mal’cev coordinates of $h$ relative to $Y'$.

The algorithm runs in space logarithmic in the input length $L$ and time quasilinear in $L$, or in $TC^0$, and the (expanded) length of each output word is bounded...
by a polynomial function of $L$. If compressed inputs are used (in $R$, $Y$, or $h$),
the space requirement is $O(M)$ and the time is $O(nM^3)$, where $n$ is the total
number of input words and $M$ bounds the size of any single input word.

Proof. Algorithm. Begin by applying Prop. 5.1 of [MMNV15] (or Lem. 5
of [MW17] in the TC$^0$ case) to compute a consistent nilpotent presentation
$G = \langle X' \mid R' \rangle$. Here $X \subset X'$, the inclusion $X \hookrightarrow X'$ induces an isomorphism
$\langle X \mid R \rangle \simeq \langle X' \mid R' \rangle$, and each element of $X' \setminus X$ is a commutator in elements of $X$. Use Subgroup Presentation to compute a nilpotent presentation
$\langle Y' \mid S \rangle$ for $H$. The generating set $Y' = \{h_1, \ldots, h_s\}$ is precisely the full-form sequence
for $H$. The relators have the form (2)-(4), and we encode the exponents appearing on the right sides in binary. To obtain $\phi$, note that each element of $Y'$
has the form $x_1^{\alpha_1} \cdots x_m^{\alpha_m}$, where $X' = \{x_1, \ldots, x_m\}$. We replace each $x_i$ with
its definition as a commutator of elements of $X$ and encode the exponents $\alpha_i$
using binary numbers. Finally, use Subgroup Membership with input $h$ and
$\{h_1, \ldots, h_s\}$, which returns an expression $h = h_{\gamma_1} \cdots h_{\gamma_s}$, giving the Mal’cev coordinates $(\gamma_1, \ldots, \gamma_s)$.

Complexity. Follows immediately from [MMNV15] and [MW17]. Note that $s$ is a constant depending on $c$ and $r$. \hfill $\Box$

We will often use this lemma in the case $Y = X$ to convert from an arbitrary
presentation of $G$ to a nilpotent presentation. In this case, we may assume the
algorithm uses $Y' = X' \supset X$. We convert all input words into their Mal’cev coordinates (relative to $X'$) at the same time, and perform further computations
directly on the Mal’cev coordinates.

Using binary numbers in the output is necessary in order to obtain quasilinear
time, since writing down a word in its expanded form takes as many steps as
the length of the word itself, which in this case is only bounded by a polynomial
function of $L$.

3 Algorithmic problems

Before presenting the algorithms, let us make a few remarks regarding their complexity analysis. The analysis of most of the algorithms is similar, so we
present here a general argument and fill in any additional details in the proof
of each algorithm.

First, note that the nilpotency class $c$ and maximum number of generators $r$
of the input group(s) are constant. All other constants are expressible in terms
of $c$ and $r$.

At the beginning of each algorithm, we convert to a nilpotent presentation,
if necessary, using Lemma 3. We denote the resulting Mal’cev basis by

$\{a_1, \ldots, a_m\}$.

Note that $m$ is constant. Word lengths are unchanged during this conversion
(see Lemma 3). We are guaranteed by [MMNV15] Thm. 2.3 that a word of
length $L$ has a Mal’cev form of length polynomial in $L$, hence its coordinates require $O(\log L)$ bits to record.

Our algorithms generally consist of a sequence of subroutine calls, using the algorithms described in §2.4 as well as those described in this section, with some minor additional processing. The complexity bounds described in §2.4 also apply to the algorithms we describe in this section, as we will see. In all cases, we prove that the total number of subroutine calls and the total number of words that must be stored in memory at any given time is constant. Consequently, the entire algorithm can, in principle, be expressed as a composition of a constant number of functions. Each such function is TC$^0$-computable, hence so is the composition. Note that to ‘store $x$ in memory’ in TC$^0$ terms means to add a parallel computation branch computing $x$.

Though it follows immediately that we have logspace solutions to these problems, we wish to prove that one may in fact run the algorithms on a logspace transducer in quasilinear time. To do so, we must show that each subroutine may be run directly ‘in memory’ on the logspace transducer.

This is achieved by invoking each subroutine in its ‘compressed’ form. Initially, all input words are converted into $O(\log L)$-bit Mal’cev coordinate form. In this process, any variable-sized set of words (subgroup generators or group relators) is reduced to a constant-sized set (the full-form sequence). This size is bounded by $m$, and we often assume it is precisely $m$ for notational convenience. Each subroutine is then called with a constant number of $O(\log L)$-bit words. It will therefore run in space $O(\log L)$ and time $O(\log^3 L)$, and produce a constant number of $O(\log n)$-bit output words.

For compressed inputs, the argument is similar. As we observed earlier, the polynomial length bound implies that the compressed size of words remains $O(M)$ throughout the algorithm. Each subroutine therefore has space complexity $O(M)$ and time complexity $O(M^3)$, so the overall space and time complexities are $O(M)$ and $O(nM^3)$.

Finally, let us note that if we have a constant number elements $g_1, \ldots, g_t$ in Mal’cev form we can, by [MMNV15] Lem. 2.10, compute the Mal’cev form of the product $g_1 \cdots g_t$ within the space and time bounds specified above, in both compressed and uncompressed cases. We use this without mention to maintain elements in coordinate form.

### 3.1 Subgroup conjugacy and normalizers

In this section we give an algorithm to determine whether or not two subgroups of a nilpotent group are conjugate and if so to compute a conjugating element. A natural by-product of this algorithm is the computation of subgroup normalizers.

We begin with a preliminary lemma solving the simultaneous conjugacy problem for tuples of commuting elements. In fact, commutation is not required, but we will obtain this stronger result (Theorem 7) as a corollary of the more complicated coset intersection algorithm.
Lemma 4. Fix positive integers $c$, $r$, and $l$. There is an algorithm that, given a finitely generated nilpotent group $G = \langle X \mid R \rangle$ of nilpotency class at most $c$ with $|X| \leq r$ and two tuples of elements $(a_1, \ldots, a_l)$ and $(b_1, \ldots, b_l)$ such that $[a_i, a_j] = [b_i, b_j] = 1$ for all $1 \leq i, j \leq l$, decides if there exists $g \in G$ such that

$$a_i^g = b_i$$

for all $1 \leq i \leq l$. The algorithm produces $g$ if one exists, returns a generating set for the centralizer of $\{b_1, \ldots, b_l\}$, and may be run in space logarithmic in the length $L$ of the input and time quasilinear in $L$, or in $\text{TC}^0$. The length of each output word is bounded by a polynomial function of $L$. If compressed inputs are used, the algorithm uses space $O(M)$ and time $O(nM^3)$, where $n = |R|$ and $M$ bounds the encoded size of each input word.

Proof. Algorithm. If necessary, use Lemma 3 to convert to a nilpotent presentation. Next, we check conjugacy of $a_1$ with $b_1$ using the Conjugacy Algorithm. If they are not conjugate, we may return ‘No’. Otherwise, we obtain $h$ such that $a_1^h = b_1$ and we compute a generating set for $C_G(b_1)$ using the Centralizer Algorithm.

If $l > 1$, we proceed recursively. Notice that $g$ exists if and only if there exists $x \in G$ such that $(a_i^h)^x = b_i$ for $i = 1, \ldots, l$, since we may put $x = h^{-1}g$. Further, such $x$ must lie in $C_G(b_1)$ since $b_1 = (a_i^h)^x = b_i^x$. Therefore it suffices to call Lemma 4 recursively with the (commuting) tuples $(a_2^h, \ldots, a_l^h)$ and $(b_2, \ldots, b_l)$ and the subgroup $C_G(b_1)$ in place of $G$. Before making the recursive call, we use Lemma 3 to convert to a nilpotent presentation for $C_G(b_1)$ and write each of $a_i^h, b_i$ relative to this presentation.

If we obtain a conjugator $x$, we may return $g = hx$, using the map $\phi$ provided by Lemma 3 to write $x$ in the original generators $X$. In addition, we obtain a generating set for the centralizer of $\{b_2, \ldots, b_l\}$ in $C_G(b_1)$, which is precisely a generating set for the centralizer of the complete set $\{b_1, \ldots, b_l\}$ in $G$. As above, we must use $\phi$ to write these words in generators $X$. If the recursive call returns ‘No’, then the tuples are not conjugate.

Complexity. The depth $l$ of the recursion is constant and we need only store $h$ and the (constant-sized) generating set for the centralizer at each step of the recursion, hence the general argument given at the beginning of the section applies.

We now give the algorithm for determining conjugacy of two subgroups.

Theorem 5. Fix integers $c$ and $r$. There is an algorithm that, given a finitely presented nilpotent group $G = \langle X \mid R \rangle$ of nilpotency class at most $c$ with $|X| \leq r$ and two subgroups $H$ and $K$, determines if there exists $g \in G$ such that

$$H^g = K$$

and if so finds such an element $g$ as well as

a generating set for the normalizer $N_G(K)$.
of $K$. The algorithm runs in space logarithmic in the total length $L$ of the input and time quasilinear in $L$, or in $T\mathcal{O}^3$, and the length of every output word is bounded by a polynomial function of $L$. If compressed inputs are used, the space complexity is $O(M)$ and the time complexity $O(nM^3)$ where $n$ is the total number of input words and $M$ bounds the encoded size of each input word.

Proof. Algorithm. Begin by converting, if necessary, to a nilpotent presentation of $G$ using Lemma 3

The algorithm recurses on the maximum $j$ such that $H \cap \Gamma_j \neq 1$ and $H \cap \Gamma_{j+1} = 1$. To find $j$, simply compute the Full-form Sequence for $H$ and observe that if the last element of the sequence begins with the letter $a_k$ then $j$ is the unique index such that $a_k\Gamma_{j+1}$ belongs to the generating set of $\Gamma_j/\Gamma_{j+1}$ (see Lemma 2). Compute similarly the maximum $j'$ such that $K \cap \Gamma_{j'} \neq 1$ and $K \cap \Gamma_{j'+1} = 1$. If $j \neq j'$, then $H$ and $K$ are not conjugate since their conjugacy would imply conjugacy of $H \cap \Gamma_i$ with $K \cap \Gamma_i$ for all $i$ (since the $\Gamma_i$ are normal subgroups), hence equality of $j$ and $j'$.

Denote $H_j = H \cap \Gamma_j$ and produce the full-form sequence for this group by taking the elements of the full-form sequence for $H$ that are in $\Gamma_j$. Proceed similarly for $K_j = K \cap \Gamma_j$. Next, we check conjugacy of $H_j$ with $K_j$.

Conjugacy of $H_j$ with $K_j$. Let $\pi : G \to G/\Gamma_{j+1}$ be the natural homomorphism. By the definition of central series, $G$ acts trivially by conjugation on $\Gamma_j$. Hence if $H_j$ and $K_j$ are conjugate then $\pi H_j = \pi K_j$. We first check if $\pi H_j = \pi K_j$, returning ‘No’ if not. To do so, it suffices to compute the Full-form Sequences for $H_j$ and $K_j$, and check them for equality.

Let $(h_1, \ldots, h_l)$ be the full-form sequence for $H_j$, computed above. We now produce a generating set $(k_1, \ldots, k_l)$ for $K_j$ such that $\overline{h_i} = \overline{k_i}$ for all $i$, as follows. Use the Preimage algorithm, with the subgroup $K_j$, the homomorphism $\pi : K_j \to G/\Gamma_{j+1}$, and the element $\overline{h_i}$, to produce each $k_i$. Since $K_j \cap \Gamma_{j+1} = 1$, $(k_1, \ldots, k_l)$ generates $K_j$.

We claim for any $x \in G$, $H_j x \pi = K_j$ if and only if $h_i^x = k_i$ for $i = 1, \ldots, l$. Indeed, since the tuples $(h_1, \ldots, h_l)$ and $(k_1, \ldots, k_l)$ are generating sets their conjugacy implies $H_j$ and $K_j$ are conjugate. Conversely, if $H_j x \pi = K_j$ then $h_i^x \in K_j$ for all $i$. But $\overline{h_i^x} = \overline{h_i} = \overline{k_i}$, and since $\pi$ is injective on $K_j$, we have $h_i^x = k_i$ for all $i$. Also observe that $H_j$ is abelian, since

$$[H_j, H_j] \leq H_j \cap \Gamma_{2j} \leq H_j \cap \Gamma_{j+1} = 1,$$

and similarly for $K_j$. Hence $(h_1, \ldots, h_l)$ and $(k_1, \ldots, k_l)$ are both tuples of commuting elements. So to determine conjugacy of $H_j$ with $K_j$ it suffices to use the algorithm of Lemma 4 to determine conjugacy of $(h_1, \ldots, h_l)$ and $(k_1, \ldots, k_l)$ and if so find a conjugator $x$ and a generating set $Y$ for $C_G(K_j)$. In fact, $C_G(K_j) = N_G(K_j)$ since if any element $y \in G$ normalizes $K_j$, then for each $i = 1, \ldots, l$ we have $k_i^y \in K_j$ and hence $k_i^y = k_i$, arguing as above.

Recursion. If $j = 0$, then $H = H_j$ and $K = K_j$ and we have already solved the problem. Otherwise, letting

$$\overline{\cdot} : N_G(K_j) \to N_G(K_j)/K_j$$
be the canonical homomorphism, we reduce the problem to conjugation of \( \hat{H}^x \) and \( \hat{K} \) in \( N_G(K_j)/K_j \), as follows.

An element \( g \) such that \( H^g = K \) exists if and only if there exists \( y \in G \) such that \( (H^x)^y = K \). Such an element \( y \) must lie in \( N_G(K_j) \), since

\[
K_j^y = (H_j^x)^y = (H \cap \Gamma_j)^{x}y = H^{xy} \cap \Gamma_j^{xy} = K \cap \Gamma_j = K_j.
\]

Now \( K \leq N_G(K_j) \), and \( H^x \leq N_G(K_j) \) since

\[
K_j = H_j^x = (H \cap \Gamma_j)^x = H^x \cap \Gamma_j \leq H^x.
\]

Finally, if \( (\hat{H}^x)^y = \hat{K} \) for some \( \hat{y} \in N_G(K_j)/K_j \), we claim that \( (H^x)^y = K \). Indeed, if \( k \in K \) then for some \( h \in H \) and \( k' \in K_j \) we have \( k = y^{-1}h^xyk' = y^{-1}(h^x(k')^{y^{-1}})y \). But \( y \in N_G(K_j) \) and \( K_j \leq H^x \), so \( h^x(k')^{y^{-1}} \in H^x \) and the inclusion \( K \subseteq H^x \) follows. The reverse inclusion is proved similarly.

In order to solve the conjugation problem of \( \hat{H}^x \) and \( \hat{K} \) in \( N_G(K_j)/K_j \), we first use Lemma 3 with the generating set \( Y \), to find a nilpotent presentation for \( N_G(K_j) \) and to convert the generating sets for \( H^x \), \( K \), and \( K_j \) into coordinate form in this presentation. Add the generators of \( K_j \) to this presentation to obtain a presentation for \( N_G(K_j)/K_j \), and call Theorem 5 with this presentation and the subgroups \( \hat{H}^x \) and \( \hat{K} \).

It is essential to prove that the value of \( j \) decreases in the recursive call. Letting \( N_j \) denote term \( j \) of the lower central series of \( N_G(K_j) \), we have that \( N_j \leq \Gamma_j \), hence \( K \cap N_j \leq K \cap \Gamma_j = K_j \), and the intersection is trivial modulo \( K_j \), hence \( j \) must decrease.

The recursive call either proves that \( \hat{H}^x \) and \( \hat{K} \) are not conjugate, in which case \( H \) and \( K \) are not conjugate, or returns a conjugator \( yK_j \) and a generating set \( Z \cdot K_j \) for the normalizer of \( \hat{K} \) in \( N_G(K_j)/K_j \). Note that \( y \) (and each element of \( Z \)) is given as a word over the generating set of \( N_G(K_j) \) with binary exponents. We convert back to the generating set \( X \) of \( G \) using the map \( \phi \) provided by Lemma 3. For the conjugator, we return the word \( g = xy \).

For the normalizer, we append to \( Z \) a generating set of \( K_j \) to obtain a generating set \( Z' \) for the normalizer of \( K \) in \( N_G(K_j) \). But this is precisely the normalizer of \( K \) in \( G \): if \( K^z = K \) for some \( z \in G \) then \( K_j^z = K^z \cap \Gamma_j = K \cap \Gamma_j = K_j \) and so \( N_G(K) \leq N_G(K_j) \).

**Complexity.** The depth of the recursion is bounded by the constant \( c \), and the number of words to store in memory is constant. \( \square \)

It should be noted that while the algorithm does not compute the normalizer of \( K \) in the event that \( H \) and \( K \) are not conjugate, one may of course obtain it by running the algorithm with \( H = K \).

### 3.2 Coset intersection

We describe an algorithm to compute the intersection of cosets in finitely generated nilpotent groups, and apply it to solving the simultaneous conjugacy
problem. Recall that in any group, the intersection $g_1H \cap g_2K$ of two cosets is, if non-empty, a coset of the intersection $H \cap K$.

**Theorem 6.** Fix integers $c$ and $r$. There is an algorithm that, given a finitely presented nilpotent group $G = \langle X \mid R \rangle$ of nilpotency class at most $c$ with $|X| \leq r$, two subgroups $H$ and $K$ of $G$, and two elements $g_1$ and $g_2$ of $G$, determines if the intersection $g_1H \cap g_2K$ is non-empty and if so, produces a generating set for $H \cap K$ and an element $g' \in g_1H \cap g_2K$, hence

$$g_1H \cap g_2K = g'(H \cap K).$$

The algorithm runs in space logarithmic in the length $L$ of the input and time quasilinear in $L$, or in $TC^0$. If compressed inputs are used, the space complexity is $O(M)$ and the time complexity $O(nM^3)$ where $n$ is the total number of input words and $M$ bounds the encoded size of each input word.

**Proof.** Begin by using Lemma 3 to convert to a nilpotent presentation for $G$, if necessary. We proceed by induction on the nilpotency class $c$.

**Base case** $c = 1$. In this case, $G$ is abelian. First, we will determine if the intersection is non-empty and if so find $g$. Writing

$$g_1H \cap g_2K = g_2(g_2^{-1}g_1H \cap K),$$

it suffices to determine if there exists $h \in H$ such that $g_2^{-1}g_1h \in K$. Since $G$ is abelian, this occurs if and only if $g_2^{-1}g_1 \in \langle H \cup K \rangle$. We use the Membership algorithm, with the union of the Full-form sequences of $H$ and $K$ as a generating set for $\langle H \cup K \rangle$, to determine if this is the case, returning ‘No’ if it is not. Otherwise, we obtain an expression of $g_2^{-1}g_1$ as a linear combination of the elements of the full-form sequence for $\langle H \cup K \rangle$. We can convert to an expression in terms of the full-form sequences for $H$ and $K$, thus obtaining an expression $g_2^{-1}g_1 = hk$ for some elements $h \in H$ and $k \in K$, by following the procedure described in Cor. 3.9 of [MMNV15] (essentially, recording an expression of each matrix row in terms of the given generators during the matrix reduction process). This corollary gives only polynomial time, but Thm. 14 of [MW17] gives the corresponding result for $TC^0$ (hence logspace), though we need the fact that $g_2^{-1}g_1$ and the full-form sequences of $H$ and $K$ are stored using only $O(|\log L|)$ bits. We now set $g = g_1h^{-1}$ and obtain $g_1H \cap g_2K = g(H \cap K)$.

We will now find a generating set for $H \cap K$. Let $\{u_1, \ldots, u_n\}$ be the generating set for $H$ and consider the homomorphism $\phi : \mathbb{Z}^n \rightarrow G$ defined by

$$\phi(\alpha_1, \ldots, \alpha_n) = u_1^{\alpha_1} \cdots u_n^{\alpha_n}$$

and the composition $\phi' : \mathbb{Z}^n \rightarrow G \rightarrow G/K$. An element $u_1^{\alpha_1} \cdots u_n^{\alpha_n}$ of $H$ is also an element of $K$ if and only if $(\alpha_1, \ldots, \alpha_n)$ is in the kernel of $\phi'$, hence $H \cap K = \phi(\ker \phi')$. To compute the kernel, add the generators of $K$ to the relators of $G$ to obtain a presentation of $G/K$, and pass this group together with $\phi'$ and the standard presentation of $\mathbb{Z}^n$ to the Kernel algorithm. Applying
\( \phi \) to each resulting subgroup generator, we obtain a generating set (in fact, the full-form sequence) for \( H \cap K \).

**Inductive case.** Denote by \( \longrightarrow : G \to G/\Gamma_c \) the canonical homomorphism. Invoke Theorem 6 recursively in \( G/\Gamma_c \) with inputs \( \overline{\Pi}, \overline{K}, \overline{\Pi}, \) and \( \overline{G} \). Note that it suffices to erase all generators of \( \Gamma_c \) to compute \( \longrightarrow \) (more formally, one may use Lemma 20).

If the recursive call determines that \( \overline{\Pi} \cap \overline{G} \) is empty, then so is \( g_1H \cap g_2K \). Otherwise, we obtain an element \( \gamma \in \overline{g_1H} \cap \overline{g_2K} \) and a generating set \( \overline{\pi}_1, \ldots, \overline{\pi}_i \) of \( \overline{\Pi} \cap \overline{K} \), hence

\[
\overline{g_1H} \cap \overline{g_2K} = \gamma(\overline{\pi}_1, \ldots, \overline{\pi}_i).
\]

Denote by \( \Lambda \) (but do not compute) the preimage of \( \overline{\Pi} \cap \overline{K} \) under \( \longrightarrow \). We will rewrite the intersection \( g_1H \cap g_2K \) in the form

\[
g_1H \cap g_2K = (g'(\Lambda \cap H)) \cap (g'c_0(\Lambda \cap K))
\]

for certain \( g' \in G, c_0 \in \Gamma_c \) defined below. Compute a \textbf{Preimage} \( x_1 \) of \( g_1^{-1}\gamma \) in \( H \) and a \textbf{Preimage} \( x_2 \) of \( g_2^{-1}\gamma \) in \( K \). Let

\[
g' = g_1x_1.
\]

Since \( g_1x_1x_2 = \overline{\gamma} = \overline{g_1x_1} \), it follows that \( (g_1x_1)^{-1}g_2x_2 = c_0 \in \Gamma_c \).

To see that (5) holds, let \( u \) be an element of the left side. Then \( u = g_1h \) and for some \( h \in H \) and \( \overline{u} = \overline{g_1h} = \gamma' \lambda \) for some \( \lambda \in \Lambda \). Then for some \( e' \in \Gamma_c \), \( u = g_1x_1\lambda e' \) hence \( \lambda e' \in H \) since \( x_1 \in H \). Clearly \( \lambda e' \in \Lambda \), hence \( u \in g'(\Lambda \cap H) \).

Similarly \( u \in g'c_0(\Lambda \cap K) \). Conversely, any element of the right side has the form \( g'h = g_1x_1h \) for some \( h \in H \) hence is in \( g_1H \), and has the form \( g'c_0k = g_2x_2k \) for some \( k \in K \) hence is in \( g_2K \).

We will now find the full-form sequences for \( \Lambda \cap H \) and \( \Lambda \cap K \). Apply the \textbf{Preimage} algorithm to compute for each \( \overline{\pi}_i \) preimages \( u'_i \in H \) and \( v'_i \in K \). Compute a generating set \( y'_1, \ldots, y'_n \) for \( H \cap \Gamma_c \) by finding the \textbf{Full-form sequence} for \( H \) and selecting only those elements that belong to \( \Gamma_c \). Similarly, compute a generating set \( z'_1, \ldots, z'_n \) for \( K \cap \Gamma_c \). We now have

\[
\Lambda \cap H = \langle u'_1, \ldots, u'_i, y'_1, \ldots, y'_n \rangle.
\]

\[
\Lambda \cap K = \langle v'_1, \ldots, v'_i, z'_1, \ldots, z'_n \rangle.
\]

Using the generating sets above, find the \textbf{Full-form sequence} \( (v_1, \ldots, v_n, z_1, \ldots, z_s) \) for \( \Lambda \cap K \), where \( z_1 \) denotes the first generator of the sequence that lies in \( \Gamma_c \).

Likewise find the \textbf{Full-form sequence} \( (u_1, \ldots, u_n, y_1, \ldots, y_n) \) for \( \Lambda \cap H \), with \( y_1 \) being the first generator in \( \Gamma_c \). Since \( \Lambda \cap H \) and \( \Lambda \cap K \) have the same image \( \overline{\Pi} \cap \overline{K} \) under \( \longrightarrow \), it follows that \( n' = n \) and for all \( i = 1, \ldots, n \) that \( u_i = v_i c_i \) for some \( c_i \in \Gamma_c \). We now have the full-form sequences

\[
\Lambda \cap H = \langle v_1c_1, \ldots, v_nc_n, y_1, \ldots, y_n \rangle.
\]

\[
\Lambda \cap K = \langle v_1, \ldots, v_n, z_1, \ldots, z_s \rangle.
\]
The next step produces a generating set of $H \cap K$ and the element $g$. The correctness of this step is argued below. Denote $C_1 = \langle c_1, \ldots, c_n, y_1, \ldots, y_t \rangle$ and consider the intersection

$$C_1 \cap (\Lambda \cap K \cap \Gamma_c)$$

in the abelian group $\Gamma_c$. Define a homomorphism $\psi : \mathbb{Z}^{n+t} \to \Gamma_c$ by

$$\psi(\alpha_1, \ldots, \alpha_n, \beta_1, \ldots, \beta_t) = c_1^{\alpha_1} \cdots c_n^{\alpha_n} y_1^{\beta_1} \cdots y_t^{\beta_t}.$$  

Using the composition $\psi' : \mathbb{Z}^{n+t} \to \Gamma_c \to \Gamma_c/(\Lambda \cap K \cap \Gamma_c)$, we may then use the Kernel algorithm, as in the base case, to produce a finitely generated subgroup $P = \langle p_1, \ldots, p_b \rangle \leq \mathbb{Z}^{n+t}$ such that

$$C_1 \cap \Lambda \cap K \cap \Gamma_c = \psi(P).$$

The sequence $(p_1, \ldots, p_b)$ is the full-form sequence for $P$, so the corresponding matrix formed is in row-echelon form. We denote $p_i = (p_{i1}, \ldots, p_{i(n+t)})$ for $i = 1, \ldots, b$. In addition, we use the Membership algorithm, as described in the base case, to find

$$c_0' \in C_1 \cap c_0(\Lambda \cap K \cap \Gamma_c)$$

if such an element exists and to write $c_0'$ in the form

$$c_0' = c_1^{\alpha'_1} \cdots c_n^{\alpha'_n} y_1^{\beta'_1} \cdots y_t^{\beta'_t}.$$  

If $c_0'$ does not exist, we return ‘No’. Otherwise, we define

$$h = (v_1 c_1)^{\alpha'_1} \cdots (v_n c_n)^{\alpha'_n} y_1^{\beta'_1} \cdots y_t^{\beta'_t}$$

and return $g = g'h$. For the generating set of $H \cap K$, define the function (it is not, in general, a homomorphism) $\theta : \mathbb{Z}^{n+t} \to \Lambda \cap H$ by

$$\theta(\alpha_1, \ldots, \alpha_n, \beta_1, \ldots, \beta_t) = (v_1 c_1)^{\alpha_1} \cdots (v_n c_n)^{\alpha_n} y_1^{\beta_1} \cdots y_t^{\beta_t}$$

and return the Full-form sequence for the subgroup generated by the set

$$\Pi = \{\theta(p_i) \mid 1 \leq i \leq b\}.$$  

It remains to prove the correctness of the last step. First, we prove that $\Pi$ generates $H \cap K = (\Lambda \cap H) \cap (\Lambda \cap K)$. Take any $p_i = (\alpha_1, \ldots, \alpha_n, \beta_1, \ldots, \beta_t)$. Then, using the fact that $C_1 \leq \Gamma_c$ is in the center of $G$,

$$\theta(p_i) = (v_1 c_1)^{\alpha_1} \cdots (v_n c_n)^{\alpha_n} y_1^{\beta_1} \cdots y_t^{\beta_t} \quad (8)$$

and

$$\theta(p_i) = v_1^{\alpha_1} \cdots v_n^{\alpha_n} c_1^{\alpha_1} \cdots c_n^{\alpha_n} y_1^{\beta_1} \cdots y_t^{\beta_t} \quad (9)$$

Line 8 gives $\theta(p_i) \in \Lambda \cap H$ and, since $c_1^{\alpha_1} \cdots c_n^{\alpha_n} y_1^{\beta_1} \cdots y_t^{\beta_t} = \psi(p_i) \in \psi(P) \subset \Lambda \cap K$, line 9 gives $\theta(p_i) \in \Lambda \cap K$. Hence $(\Pi) \leq H \cap K$.  

16
For the opposite inclusion, let \((\Lambda \cap H)_i = \langle v_i c_i, \ldots, v_n c_n, y_1, \ldots, y_l \rangle \) for \(1 \leq i \leq n\) and \((\Lambda \cap H)_{n+1} = \langle y_1, \ldots, y_l \rangle \). We will prove, by induction on \(i\) in the reverse order \(n+1, \ldots, 1\), that
\[
(\Lambda \cap H)_i \cap (\Lambda \cap K) \leq \langle \Pi \rangle
\]
for all \(i = n + 1, \ldots, 1\) (in particular for \(i = 1\)). For the base case \(i = n + 1\), let \(q \in (\Lambda \cap H)_{n+1} \cap (\Lambda \cap K)\). Then \(q = e_{i}^{0} \cdots e_{n}^{0} y_{1}^{\beta_{1}} \cdots y_{t}^{\beta_{t}}\) for some \((0, \ldots, 0, \beta_{1}, \ldots, \beta_{t}) = p\). Since \(q \in C_{1} \cap \Lambda \cap K\), we have \(p \in \Pi\). Since the matrix corresponding to \(P\) is in row echelon form, we may write \(p\) as a linear combination
\[
p = \sum_{j=k}^{b} \gamma_{j} p_{j}
\]
where \(p_{j} = (0, \ldots, 0, p_{j(n+1)}, \ldots, p_{j(n+t)})\) for all \(j \geq k\). Then
\[
q = (y_{1}^{p_{k(n+1)}} \cdots y_{t}^{p_{k(n+t)}}) \gamma_{k} \cdots (y_{1}^{p_{k(n+1)}} \cdots y_{t}^{p_{k(n+t)}}) \gamma_{b} = \theta(p_{k})^{\gamma_{k}} \cdots \theta(p_{b})^{\gamma_{b}}
\]
hence \(q \in \langle \Pi \rangle\).

For the inductive case, assume that \((\Lambda \cap H)_{i+1} \cap (\Lambda \cap K) \leq \langle \Pi \rangle\) for some \(i + 1 \leq n + 1\) and let \(q \in (\Lambda \cap H)_{i} \cap (\Lambda \cap K)\). Then
\[
q = (v_{1} c_{1})^{0} \cdots (v_{i-1} c_{i-1})^{0} (v_{1} c_{1})^{\alpha_{i}} \cdots (v_{n} c_{n})^{\alpha_{n}} y_{1}^{\beta_{1}} \cdots y_{t}^{\beta_{t}}
\]
for some \((0, \ldots, 0, \alpha_{i}, \ldots, \alpha_{n}, \beta_{1}, \ldots, \beta_{t}) = p'\). Since \(q \in \Lambda \cap K\) and \(v_{1}^{\alpha_{1}} \cdots v_{n}^{\alpha_{n}} \in \Lambda \cap K\), it follows, rewriting \(q\) as in (4), that \(e_{i}^{\alpha_{i}} \cdots e_{n}^{\alpha_{n}} y_{1}^{\beta_{1}} \cdots y_{t}^{\beta_{t}} \in \Lambda \cap K\) and hence \(p' \in \Pi\). Hence
\[
p = \sum_{j=k'}^{b} \gamma_{j}^{'} p_{j}^{'}
\]
for some \(\gamma_{j}'\) and \(k' \geq 1\) such that \(p_{j} = (0, \ldots, 0, p_{j}, \ldots, p_{j(n+t)})\) for all \(j \geq k'\).

Now consider the element
\[
q' = q \theta(p_{k'})^{-\gamma_{k}'} \cdots \theta(p_{b})^{-\gamma_{b}'}
\]
In the word \(\theta(p_{k'})^{-\gamma_{k}'} \cdots \theta(p_{b})^{-\gamma_{b}'}\), the generators \(v_{1} c_{1}, \ldots, v_{i-1} c_{i-1}\) do not appear. Further, the total exponent sum of the generator \(v_{i} c_{i}\) is \(-\alpha_{i}\), while in \(q\) it is \(\alpha_{i}\). Since for any \(i < l \leq n\) and \(1 \leq l' \leq t\) the commutators \([v_{i} c_{i}, v_{l'} c_{l'}]\) and \([v_{i} c_{i}, y_{l'}]\) are elements of \((\Lambda \cap H)_{i+1}\) we may collect all occurrences of \(v_{i} c_{i}\) and eliminate its occurrence in \(q'\). Hence \(q' \in (\Lambda \cap H)_{i+1} \cap (\Lambda \cap K)\). By induction, \(q' \in \langle \Pi \rangle\) hence \(q \in \langle \Pi \rangle\) as well.

Regarding the intersection being non-empty, observe that by (5),
\[
g_{1} H \cap g_{2} K \neq \emptyset \iff (\Lambda \cap H) \cap c_{0}(\Lambda \cap K) \neq \emptyset.
\]
Now if \( g = (v_1c_1)^{\alpha_1} \cdots (v_nc_n)^{\alpha_n}y_1^{\beta_1} \cdots y_t^{\beta_t} \in \Lambda \cap H \) is also an element of \( c_0(\Lambda \cap K) \) then, again rewriting as in (9), \( c_1^{\alpha_1} \cdots c_n^{\alpha_n}y_1^{\beta_1} \cdots y_t^{\beta_t} \) is an element of \( C_1 \) which also lies in \( c_0(\Lambda \cap K \cap \Gamma_c) \) hence \( c'_0 \) exists. Conversely, if \( c'_0 \) exists, then we have

\[
e_0' = c_1^{\alpha_1} \cdots c_n^{\alpha_n}y_1^{\beta_1} \cdots y_t^{\beta_t} = c_0z_1^{\gamma_1} \cdots z_s^{\gamma_s}
\]

hence

\[
(v_1c_1)^{\alpha_1} \cdots (v_nc_n)^{\alpha_n}y_1^{\beta_1} \cdots y_t^{\beta_t} = c_0v_1^{\alpha_1} \cdots v_n^{\alpha_n}z_1^{\gamma_1} \cdots z_s^{\gamma_s}
\]

is an element of \( (\Lambda \cap H) \cap c_0(\Lambda \cap K) \), hence this intersection is non-empty. This proves the correctness of the decision problem.

Finally, we must show that \( g \in g_1H \cap g_2K \). Since \( h \in \Lambda \cap H \), we have \( g = g'h \in g'\Lambda \cap H \). Since \( c'_0 \in c_0(\Lambda \cap K) \), we have \( c'_0 = c_0k \) for some \( k \in \Lambda \cap K \) hence

\[
g = g'h = g'v_1^{\alpha_1} \cdots v_n^{\alpha_n} c'_0 = g' c_0(kv_1^{\alpha_1} \cdots v_n^{\alpha_n})
\]

is an element of \( g'c_0(\Lambda \cap K) \), as required.

**Complexity.** The depth of the recursion is \( c \), which is constant, so the total number of subroutine calls is constant. The total number of group elements to record is also constant. \( \square \)

As an application of the intersection algorithm, we may generalize Lemma 4 to solve the simultaneous conjugation problem for tuples in nilpotent groups.

**Theorem 7.** Fix integers \( c, r, \) and \( l \). There is an algorithm that, given a nilpotent group \( G = \langle X \mid R \rangle \) of nilpotency class at most \( c \) with \( |X| \leq r \) and two tuples \((a_1, \ldots, a_l)\) and \((b_1, \ldots, b_l)\) of elements of \( G \), computes:

- an element \( g \in G \) such that
  
  \[
g^{-1}a_ig = b_i
  \]

  for \( i = 1, \ldots, l \)

- a generating set for the centralizer \( C_G(b_1, \ldots, b_l) \), or determines that no such element \( g \) exists.

The algorithm runs in space logarithmic in the size \( L \) of the input and time quasilinear in \( L \), or in \( TC^0 \), and the length of each output word is bounded by a polynomial function of \( L \). If compressed inputs are used, the space complexity is \( O(M) \) and the time complexity \( O(nM^3) \) where \( n = |R| \) and \( M \) bounds the encoded size of each input word.

**Proof. Algorithm.** Begin by applying Lemma 4 to convert to a nilpotent presentation if necessary. Next, for each \( i = 1, \ldots, l \), use the Conjugacy algorithm to find \( g_i \in G \) such that \( a_i^{g_i} = b_i \). If any pair is not conjugate, then \( g \) does not exist and we may return 'No'. We also use the Centralizer algorithm to find, for each \( i \), a generating set for \( C_G(b_i) \).
Now for any $g \in G$ and any $i$, the equation
\[ a^g_i = a^{g_i g^{-1}_i g} = b^{g^{-1}_i g} \]
shows that $a^g_i = b_i$ if and only if $g_i^{-1} g \in C_G(b_i)$, i.e. $g \in g_i C_G(b_i)$. Hence the set of all possible conjugators is precisely the coset intersection $\bigcap_{i=1}^l g_i C_G(b_i)$ which we may compute by iterating Theorem 6. As a by-product, we obtain a generating set for
\[ \bigcap_{i=1}^l C_G(b_i) = C_G(b_1, \ldots, b_l). \]

**Complexity.** Since $l$ is fixed, the number of subroutine calls and elements of $G$ to store is constant. \qed

### 3.3 Torsion subgroup

In every nilpotent group $G$ the set $T$ consisting of all elements of finite order forms a subgroup called the torsion subgroup. We give an algorithm to compute, from a presentation of $G$, a generating set and presentation for $T$ as well as its order. We follow an algorithm outlined in [KRR+69].

**Theorem 8.** Fix positive integers $c$ and $r$. There is an algorithm that, given a finitely presented nilpotent group $G = \langle X \mid R \rangle$ of nilpotency class at most $c$ with $|X| \leq r$, produces

- a generating set for the torsion subgroup $T$ of $G$,
- a presentation for $T$, and
- the order of $T$.

The algorithm runs in space logarithmic in the size $L$ of the given presentation and time quasilinear in $L$, or in $TC^0$. The length of each output word is bounded by a polynomial function of $L$ and the number of such words is bounded by a constant. If compressed inputs are used, the space complexity is $O(nM)$ and the time complexity is $O(n M^3)$, where $n = |R|$ and $M$ bounds the length of each relator in $R$.

**Proof.** Define inductively a sequence $T_1, T_2, \ldots$ of finite normal subgroups of $G$ as follows. Let $T_1 = T(Z(G))$, which is clearly finite and normal. For $i > 1$ define the homomorphism $\phi_i : G \to G/T_{i-1}$ and set
\[ T_i = \phi_i^{-1} (T(Z(G/T_{i-1}))). \]

Since $Z(G/T_{i-1})$ is abelian and finitely generated, $T(Z(G/T_{i-1}))$ is finite and hence finiteness of $T_i$ follows from that of $T_{i-1}$. Normality of $T_i$ follows from normality of $T_{i-1}$ in $G$ and of $T(Z(G/T_{i-1}))$ in $G/T_{i-1}$. Since $G$ is Noetherian, the sequence must stabilize at some $T_k$. But then $T(Z(G/T_k))$ is trivial, hence
$G/T_k$ is torsion-free (its torsion subgroup must otherwise intersect its center), hence $T_k = T$.

Algorithm. We compute the sequence described above. Begin by applying Lemma 3 to compute a nilpotent presentation $G = \langle A \mid S \rangle$. Since $Z(G)$ is simply the centralizer of any generating set, we may find the full-form sequence $(h_1, \ldots, h_m)$ for $Z(G)$ using Theorem 7 with the set $A$. Since $Z(G)$ is abelian, its torsion subgroup is generated by the set $X_1$ consisting of elements $h_i$ such that $i \in T$. Note that $T$ is determined by examining the relators of the form $[h_i, h_j] = S$.

Now assume, by induction, that we have a generating set $X_i$ for $T_i$. Use Theorem 7 with the nilpotent group $G/T_i = \langle A \mid S \cup X_i \rangle$ and the set $A$ to find, as described in the base case, the full-form sequence $(\tau_1 T_i, \ldots, \tau_m T_i)$ for $T(Z(G/T_i))$. Then $X_i \cup \{\tau_1, \ldots, \tau_m\}$ generates $T_{i+1}$, and we compute the the Full-form sequence $X_{i+1}$ of $T_i$. If $X_{i+1} = X_i$, then $T_{i+1} = T_i = T$. Otherwise, we proceed with the next step of the induction.

Once we obtain the full-form sequence $(t_1, \ldots, t_n)$ for $T$, it suffices to run Subgroup Presentation to give a presentation for $T$. Denote by $\pi_1, \ldots, \pi_n$ the pivot columns of the matrix associated with $(t_1, \ldots, t_n)$ and by $\alpha_{ij}$ the $(i,j)$-entry of this matrix. Then every element of $T$ may be expressed uniquely in the form $t_1^{\beta_1} \cdots t_n^{\beta_n}$ where $0 \leq \beta_i < e_{\pi_i}/\alpha_{i\pi_i}$, and every such expression gives a different element. Hence the order of $T$ is

$$|T| = \prod_{i=1}^n e_{\pi_i}/\alpha_{i\pi_i}.$$ 

Complexity. First, we will prove that the depth of the recursion is bounded by $c$. Let $Z_i = \{h \in G \mid [h, g] \in Z_{i-1} \text{ for all } g \in G\}$ be the $i^{th}$ term of the upper central series of $G$, with $Z_1 = Z(G)$. We claim that $T_i \supseteq Z_i \cap T$

for all $i = 1, \ldots, c$, hence $T_c \supseteq G \cap T$ so $T_c = T$ and the depth of the recursion is bounded by $c$. We proceed by induction. For $i = 1$ we have $T_1 = T(Z(G)) = Z_1 \cap T$. Now let $g \in Z_i \cap T$ and consider $i_1(g) = \mathbf{r}$. Let $\tilde{h} \in G/T_{i-1}$ and consider $[\mathbf{r}, \tilde{h}] = [g, h]$. Since $g \in T$, we have $[g, h] = g^{-1}g \in T$ and since $g \in Z_i$, we have $[g, h] \in Z_{i-1}$. By the inductive assumption, $[g, h] \in T_{i-1}$ hence $[\mathbf{r}, \tilde{h}] = 1$ hence $\mathbf{r} \in Z(G/T_{i-1})$. Clearly $\mathbf{r} \in T(Z(G/T_{i-1}))$ hence $g \in T_i$, proving the claim.

Since the depth of the recursion is constant, the total number of subroutine calls is constant, as is the number of elements kept in memory, since each $X_i$ is a full-form sequence (hence of bounded length).

In computing the order of $T$, recall that the numbers $e_i$ where $i \in T$, appear as exponents in the nilpotent presentation computed by Lemma 3. Consequently, each is bounded by a polynomial function of $L$. Since the length $n$ of the full-form sequence for $T$ is bounded by a constant, the order of $T$ is polynomially bounded.
Corollary 9. If $G = \langle X \mid R \rangle$ is a nilpotent group of nilpotency class $c$ with $|X| \leq r$, then the order of the torsion subgroup of $G$ is bounded by a polynomial function of the sum of the lengths of the relators $R$.

3.4 Isolator

Recall that the isolator of $H$ in $G$ is defined by

$$\text{Is}_G(H) = \{g \in G \mid g^n \in H \text{ for some } n \neq 0\}$$

and, in nilpotent groups, forms a subgroup.

Theorem 10. Fix integers $c$ and $r$. There is an algorithm that, given a finitely presented nilpotent group $G = \langle X \mid R \rangle$ of nilpotency class at most $c$ with $|X| \leq r$, and a subgroup $H \leq G$, computes a generating set for the isolator $\text{Is}_G(H)$.

The algorithm runs in space logarithmic in the length $L$ of the input and time quasilinear in $L$, or in $TC^0$, and the length of each generator is bounded by a polynomial function of $L$. If compressed inputs are used, the space complexity is $O(M)$ and the time complexity $O(nM^3)$ where $n$ is the number of input words and $M$ bounds the encoded size of each input word.

Proof. Algorithm. First, apply Lemma 3 to convert to a nilpotent presentation $G = \langle a_1, \ldots, a_m \mid S \rangle$.

Let $N^0 = H$ and for $i > 0$ define $N^i = N_G(N^{i-1})$, the normalizer of $N^{i-1}$ in $G$. It is proved in [KM79] Thm. 16.2.2 that $N^c = G$. Using Theorem 5 we compute in turn the full-form generating sequences for each of the subgroups $H, N^1, \ldots, N^c$ and using the Subgroup Presentation algorithm we compute a nilpotent presentation

$$N^i = \langle X_i \mid R_i \rangle$$

for each. We now proceed, by induction, to compute for each $i = 0, \ldots, c$ a generating set $Y_i$ for $\text{Is}_{N^i}(H)$. For $i = 0$, we have $\text{Is}_{N^0}(H) = H$ and we use the computed full-form sequence $X_0$ for $H$. Now assume that we have a generating set $Y_{i-1}$ for $\text{Is}_{N^{i-1}}(H)$.

The subgroup $N^{i-1}$ is normal in $N^i$, and we will find the torsion subgroup of $N^i/N^{i-1}$. Using Lemma 8 write each element of $X_{i-1}$ in its $X_i$-coordinates. Appending these elements to $R_i$ we obtain a presentation of $N^i/N^{i-1}$, which we pass to Theorem 8 to obtain a generating set $\{\tau_1 N^{i-1}, \ldots, \tau_m N^{i-1}\}$ for the torsion subgroup. Then $\text{Is}_{N^i}(N^{i-1})$ is generated by $X_{i-1} \cup \{\tau_1, \ldots, \tau_r\}$. Converting these elements back to generators of $G$, we then compute the Full-form sequence $Z_i$ for $\text{Is}_{N^i}(N^{i-1})$ and, using Lemma 3 the corresponding nilpotent presentation

$$\text{Is}_{N^i}(N^{i-1}) = \langle Z_i \mid S_i \rangle.$$

We claim that $\text{Is}_{N^{i-1}}(H) \subseteq \text{Is}_{N^i}(N^{i-1})$. Indeed, the property that $g^n \in H$ for some $n > 0$ is unchanged under conjugation, and since $N^i$ normalizes $N^{i-1}$
all conjugates remain in $N_{i-1}$. Using Lemma 3 write each element of $Y_{i-1}$ in terms of the generators $Z_i$ and append these words to $S_i$ to obtain a presentation of $\text{Is}_{N_i}(N_{i-1})/\text{Is}_{N_{i-1}}(H)$. Now apply Theorem 8 to compute a generating set $\{\sigma_1\text{Is}_{N_{i-1}}(H), \ldots, \sigma_r\text{Is}_{N_{i-1}}(H)\}$ for the torsion subgroup.

Set $Y_i = \{\sigma_1, \ldots, \sigma_r\} \cup Y_{i-1}$, using the two prior calls to Lemma 3 to write each $\sigma_i$ in generators of $G$. We claim that $Y_i$ generates $\text{Is}_{N_i}(H)$. Clearly $Y_{i-1} \subset \text{Is}_{N_{i-1}}(H) \subset \text{Is}_{N_i}(H)$. For each $\sigma_i$ there exists $p_i > 0$ such that $\sigma_i^{p_i} \in \text{Is}_{N_{i-1}}(H)$, therefore there exists $q_i > 0$ such that $\sigma_i^{p_i q_i} \in H$. Hence $Y_i \subset \text{Is}_{N_i}(H)$. Now if $g \in \text{Is}_{N_i}(H)$ then $g^{n_i} \in H$ for some $n_i > 0$ and $g \in \text{Is}_{H_i}(H_{i-1})$, hence $g \text{Is}_{H_i-1}(H_i)$ lies in the torsion subgroup of $\text{Is}_{H_i}(H_{i-1})/\text{Is}_{H_{i-1}}(H_i)$, and so $g$ is an element of the subgroup generated by $Y_i$.

Finally, return the Full-form sequence for $Y_n$.

**Complexity.** Since the number of elements in a full-form sequence is bounded by $m$, the total number of elements in the sequences for $N^i$, $i = 0, \ldots, c$, is constant, as is the number of relators in each $R_i$ and the number of elements in the generating set for the torsion subgroups. The depth of the recursion is bounded by $c$. Hence the total number of elements to store and the number of subroutine calls is constant.

### References

[BMO16] Gilbert Baumslag, Charles F. Miller, and Gretchen Ostheimer. Decomposability of finitely generated torsion-free nilpotent groups. *Internat. J. Algebra Comput.*, 26(8):1529–1546, 2016.

[DLS15] Moon Duchin, Hao Liang, and Michael Shapiro. Equations in nilpotent groups. *Proc. Amer. Math. Soc.*, 143(11):4723–4731, 2015.

[GMO16a] Albert Garreta, Alexei Miasnikov, and Denis Ovchinnikov. Properties of random nilpotent groups. 2016. Preprint. arXiv:1612.01242 [math.GR].

[GMO16b] Albert Garreta, Alexei Miasnikov, and Denis Ovchinnikov. Random nilpotent groups, polycyclic presentations, and diophantine problems. 2016. arXiv:1612.02651 [math.GR].

[GS80] Fritz Grunewald and Daniel Segal. Some general algorithms. II. Nilpotent groups. *Ann. of Math. (2)*, 112(3):585–617, 1980.

[HEO05] Derek F. Holt, Bettina Eick, and Eamonn A. O’Brien. *Handbook of computational group theory*. Discrete Mathematics and its Applications (Boca Raton). Chapman & Hall/CRC, Boca Raton, FL, 2005.

[KLZ15] Daniel König, Markus Lohrey, and Georg Zetzsche. Knapsack and subset sum problems in nilpotent, polycyclic, and co-context-free groups. 2015. arXiv:1507.05145 [math.GR].

22
[KM79] M. I. Kargapolov and Ju. I. Merzljakov. Fundamentals of the theory of groups, volume 62 of Graduate Texts in Mathematics. Springer-Verlag, New York-Berlin, 1979. Translated from the second Russian edition by Robert G. Burns.

[KRR+69] M. I. Kargapolov, V. N. Remeslennikov, N. S. Romanovskii, V. A. Roman’kov, and V. A. Čurkin. Algorithmic questions for $\sigma$-powered groups. Algebra i Logika, 8:643–659, 1969.

[Loh12] Markus Lohrey. Algorithms on SLP-compressed strings: a survey. Groups Complex. Cryptol., 4(2):241–299, 2012.

[Loh14] Markus Lohrey. The compressed word problem for groups. Springer-Briefs in Mathematics. Springer, New York, 2014.

[Loh15] Markus Lohrey. Rational subsets of unitriangular groups. Internat. J. Algebra Comput., 25(1-2):113–121, 2015.

[MMNV15] Jeremy Macdonald, Alexei Miasnikov, Andrey Nikolaev, and Svetla Vassileva. Logspace and compressed-word computations in nilpotent groups. 2015. Preprint. arXiv:1503.03888 [math.GR].

[MT16] Alexei Mishchenko and Alexander Treier. Knapsack problem for nilpotent groups. 2016. Preprint. arXiv:1606.08584 [math.GR].

[MW17] Alexei Miasnikov and Armin Weiβ. $TC^0$ circuits for algorithmic problems in nilpotent groups. 2017. To appear in MFCS 2017 proceedings. Preprint at arXiv:1702.06616 [math.GR].

[Rom77] V. A. Roman’kov. Unsolvability of the problem of endomorphic reducibility in free nilpotent groups and in free rings. Algebra i Logika, 16(4):457–471, 494, 1977.

[Rom16] Vitaly A. Roman’kov. Diophantine questions in the class of finitely generated nilpotent groups. J. Group Theory, 19(3):497–514, 2016.

[Sim94] Charles C. Sims. Computation with finitely presented groups, volume 48 of Encyclopedia of Mathematics and its Applications. Cambridge University Press, Cambridge, 1994.

[Vol99] Heribert Vollmer. Introduction to circuit complexity. Texts in Theoretical Computer Science. An EATCS Series. Springer-Verlag, Berlin, 1999. A uniform approach.